Calhoun
Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

Theses and Dissertations | Thesis Collection

2010-12

Software system architecture modeling methodology for naval gun weapon systems

Rivera, Joey

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/10504

DUDLEY KNOX LIBRARY

http://www.nps.edu/library

Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

SOFTWARE SYSTEM ARCHITECTURE MODELING METHODOLOGY FOR NAVAL GUN WEAPON SYSTEMS

by

Joey Rivera

December 2010

Dissertation Supervisor: Mikhail Auguston

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** December 2010	**3. REPORT TYPE AND DATES COVERED** Dissertation	
4. TITLE AND SUBTITLE: Software System Architecture Modeling Methodology for Naval Gun Weapon Systems		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Joey Rivera			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number: N/A.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

This dissertation describes the development of an architectural modeling methodology that supports the Navy's requirement to evaluate potential changes to gun weapon systems in order to identify potential software safety risks. The modeling methodology includes a tool ("Eagle6") that is based on the Monterey Phoenix (MP) modeling methodology, and has the capability to create and verify MP models, execute formal assertions via pre-defined macro commands, and a visualization tool that generates graphical representations of model scenarios. The Eagle6 toolset has two scenario generation modes, Exhaustive Search for model verification within scope, and Random trace generation for statistical estimates of nonfunctional properties, such as performance. The dissertation demonstrates how the Eagle6 tool may improve the SSSTRP evaluation process by including a methodology to use formal assertions to test for software states that are considered unsafe.

14. SUBJECT TERMS Open Architecture, Software Requirements, Software Safety, COTS Safety Analysis, Software System Architecture, Modeling, Environmental Modeling, Assertion Checking			**15. NUMBER OF PAGES** 195
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SOFTWARE SYSTEM ARCHITECTURE MODELING METHODOLOGY FOR NAVAL GUN WEAPON SYSTEMS

Joey Rivera
Major, United States Army Reserve
BGS, Indiana University, 1993
M.A., Webster University, 2004

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2010

Author: _____
Joey Rivera

Approved by:

_____	_____
Mikhail Auguston	Thomas V. Huynh
Computer Science	Systems Engineering
Dissertation Committee Chair	Co-Advisor
_____	_____
Ronald Finkbine	Robert Harney
Computer Science	Systems Engineering
Indiana University Southeast	
_____	_____
Peter Musial	Clifford Whitcomb
Systems Engineering	Systems Engineering

Approved by: _____
Peter Denning, Chairman, Department of Computer Science

Approved by: _____
Douglas Moses, Associate Provost for Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This dissertation describes the development of an architectural modeling methodology that supports the Navy's requirement to evaluate potential changes to gun weapon systems in order to identify potential software safety risks. The modeling methodology includes a tool (Eagle6) that is based on the Monterey Phoenix (MP) modeling methodology, and has the capability to create and verify MP models, execute formal assertions via pre-defined macro commands, and a visualization tool that generates graphical representations of model scenarios. The Eagle6 toolset has two scenario generation modes, Exhaustive Search for model verification within scope, and Random trace generation for statistical estimates of nonfunctional properties, such as performance. The dissertation demonstrates how the Eagle6 tool may improve the SSSTRP evaluation process by including a methodology to use formal assertions to test for software states that are considered unsafe.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

ACAT	Acquisition Category
CA	Code Analysis
CE/D	Concept Exploration and Definition
CINCLANTFLT	Commander in Chief Atlantic Fleet
CINCPACFLT	Commander in Chief Pacific Fleet
CMC	Commandant Marine Corps
CNO	Chief of Naval Operations
COMNAVAIRSYSCOM	Commander, Naval Air Systems Command
COMNAVORDCEN	Commander, Naval Ordnance Center
COMMARCORSYSCOM	Commander, Marine Corps Systems Command
COMOPTEVFOR	Commander, Operational Test Evaluation Force
ConOps	Concept of Operations
DDA	Detailed Design Analysis
DON	Department of the Navy
DRPM	Direct Reporting Program Manager
ECP	Engineering Change Proposal
ESD	Electrostatic Discharge
EMD	Engineering and Manufacturing Development
EOD	Explosive Ordnance Disposal
FOC	Full Operational Capability
FHA	Functional Hazard Analysis
FMECA	Failure Modes, Effects and Criticality Analysis
FTA	Fault Tree Analysis
GWS	Gun Weapon System
HAR	Hazard Action Report
HERO	Hazards of Electromagnetic Radiation to Ordnance
IOC	Initial Operational Capability

IPS	Integrated Program Summary
LRIP	Low Rate Initial Production
MARFORLANT	Marine Forces Atlantic
MARFORPAC	Marine Forces Pacific
MDA	Milestones Decision Authority
MP	Monterey Phoenix
MPS	Maritime Prepositioning Ship
NAVSURFWARCENDIV	Naval Surface Warfare Center Division
NDI	Non-Development Item
OPEVAL	Operational Evaluation
ORDALTS	Ordnance Alterations
O&SHA	Operating and Support Hazard Analysis
PDA	Preliminary Design Analysis
PEO	Program Executive Officer
PHA	Preliminary Hazard Analysis
PHL	Preliminary Hazard List
PHST	Packaging, Handling, Storage and Transportation
PIP	Product Improvement Program
PM	Program Manager
POP	Performance Oriented Packaging
SAR	Safety Assessment Report
SHA & SSHA	System and Sub-System Hazard Analyses
SHIPALTS	Ship Alterations
SOF	Special Operations Forces
SRA	Software Requirements Analysis
SSSTRP	Software System Safety Technical Review Panel
SSWG	System Safety Working Group
STRA	Software Test Results Analysis
TECHEVAL	Technical Evaluation

TDP	Technical Data Package
TEMP	Test and Evaluation Master Plan
TRP	Technical Review Panel
USSOCOM	United States Special Operations Command
VERTREP	Vertical Replenishment
WSESRB	Weapon System Explosives Safety Review Board

EXECUTIVE SUMMARY

The U.S. Navy uses the Weapons System Explosive Safety Review Board (WSESRB pronounced *"we-serb"*) to evaluate potential changes to weaponry systems on naval ships. The WSESRB has a Software System Safety Review Panel (SSSTRP—pronounced "*sis-trip"*) subcommittee that focuses on the software safety aspects of weapon system changes. The SSSTRP process evaluates potential software systems during the pre-acquisition process, and reports the findings to the WSESRB. The SSSTRP community is experiencing a high vendor failure rate that results in delays to the acquisition process, and delays to equipment upgrades that lead to an improved war fighting capability. This dissertation is the result of researching three years of SSSTRP reports, and determining the causes of vendors failing the SSSTRP process. It also includes recommendations for improved SSSTRP processes, and tools that accompany the process improvements.

The SSSTRP process improvements within this dissertation center around a modeling and simulation tool named "Eagle6." Eagle6 is a web-based application that provides the SSSTRP community the ability to test the effects of potential weapon system architectural changes on the existing legacy system. Eagle6 uses formal methods to create macro queries that enable the nontechnical user to test both functional and nonfunctional system and software requirements, while generating reports that are understandable by both technical and nontechnical SSSTRP members. The tool is publically available on the web at www.Eagle6.com, and includes tutorials and sample models.

ACKNOWLEDGMENTS

There are many people who have contributed to this research effort, both directly and indirectly. First and foremost, I would like to thank my wife, Beth, for being so supportive during this endeavor. I also thank my advisor, Dr. Auguston, for his commitment to "teach" during my research, for his sound professional and personal advice, and for being such an advocate for my research during my tenure at NPS. You are, without a doubt, the best teacher I have ever known. Thanks to Dr. Huynh for his structure, guidance, and patience over the last year. Thanks to Alex and Michael Gociu for their help on the Eagle6 application programming. Thanks to Paul Dailey for his encouragement and assistance in studying for exams. Finally, thanks to all my friends and family who encouraged me when I needed it most.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The objective of the research was to identify the problems associated with the high number of software safety failures associated with the Navy's software system acquisition process. Software Safety is defined as "The software has unintended (and unsafe) behavior beyond what is specified in the requirements" (Leveson, 1995). This dissertation includes research on three years of unclassified software system safety evaluation reports and an analysis of the findings (Chapter II). A prototype modeling methodology, and the ability to apply the modeling methodology to the software safety domain, is demonstrated in Chapter III. The contributions of this research are as follows:

- A prototype methodology and tools to support software system safety analysis for the Navy's software system acquisition process
- Higher fidelity of software system safety evaluation using tools that support assertion checking
- Two methods for architecture testing using exhaustive search for model verification, and random scenario generation for statistical estimates of nonfunctional requirements, such as performance
- Extension of Monterey Phoenix Modeling Methodology to include a framework that uses predefines macro queries to execute aggregate operations over events

A. PROBLEM OVERVIEW

Chapter I contains information describing the Navy's Weapon System Explosive Safety Review Board certification process for Software Systems. Specifically, this chapter describes the SSSTRP evaluation process, and the impact to naval operations of the vendor failing the SSSTRP evaluation process. The purpose of this chapter is to describe the process and results related to determining the causal factors for vendors failing the SSSTRP evaluation

process. The results of this research demonstrate that the SSSTRP evaluation process lacks sufficient software safety evaluation methodology and tools.

B. INTRODUCTION TO THE PROBLEM

The United States Navy formed the Weapon System Explosives Safety Review Board (WSESRB) in 1968 as a result of a fire on the USS Forrestal (CV-59) (U.S. Navy, 2007). The subsequent investigation recommended the establishment of an independent review process (Naval Sea Systems Command, 1997). The report highlighted the need to ensure that safety requirements for explosives were met for all munitions introduced to the Fleet.

The WSESRB's responsibility is to review the overall safety aspects of each weapon system, explosive system, and related system to ensure that weapon system safety requirements are in compliance. After assessing the degree of compliance with existing criteria, the WSESRB provides a recommendation to the program manager, program sponsor, Chief of Naval Operations (CNO), and the Milestone Decision Authority (MDA) on the adequacy of the safety program and on whether the proposed weapon system should advance to the next stage in the acquisition cycle. At the discretion of the WSESRB Chairperson, special WSESRB Technical Review Panels (TRPs) may review specific safety aspects requiring special expertise (e.g., ordnance-related software safety) in weapon systems. An appointed TRP Chairperson leads the TRP team that has at least two other members. Naval Systems Commanders, upon request from the WSESRB Chairperson, may identify a member to serve on TRPs. These members are subject-matter experts and have expertise in the applicable area of the TRP. Other members and technical advisors, chosen for their expertise, are appointed at the discretion of the TRP Chairperson.

Recommendations made by TRPs are presented to the Program Office and the WSESRB at the conclusion of the TRP meeting; however, the TRP recommendations do not become official until the WSERSRB reviews and endorses the results. The WSESRB may accept, modify, or reject the

recommendations of the TRP. The results of the WSESRB action on the TRP recommendations are provided to the Program Office.

Dahlgren Division, Naval Surface Warfare Center (NAVSURFWARCENDIV Dahlgren), Dahlgren, Virginia, acts as a principal activity for system safety support to the WSESRB, as well as chairing the ordnance-related Software Systems Safety Technical Review Panel (SSSTRP) and other TRPs as assigned. The evaluation process contains: (1) developing and recommending, with WSESRB approval, TRP review criteria, and project data; (2) coordinating meetings of the SSSTRP with members and program offices; (3) assisting the program office in tailoring TRP review criteria for the type of program and the current program phase; (4) identifying qualified technical advisors to participate in the TRP, and, with the WSESRB chairperson's concurrence, arranging for their participation; (5) scheduling meetings of the TRP at the request of the WSESRB chairperson; and (6) providing a summary report of the TRP findings and recommendations of the SSSTRP TRP to the full WSESRB.

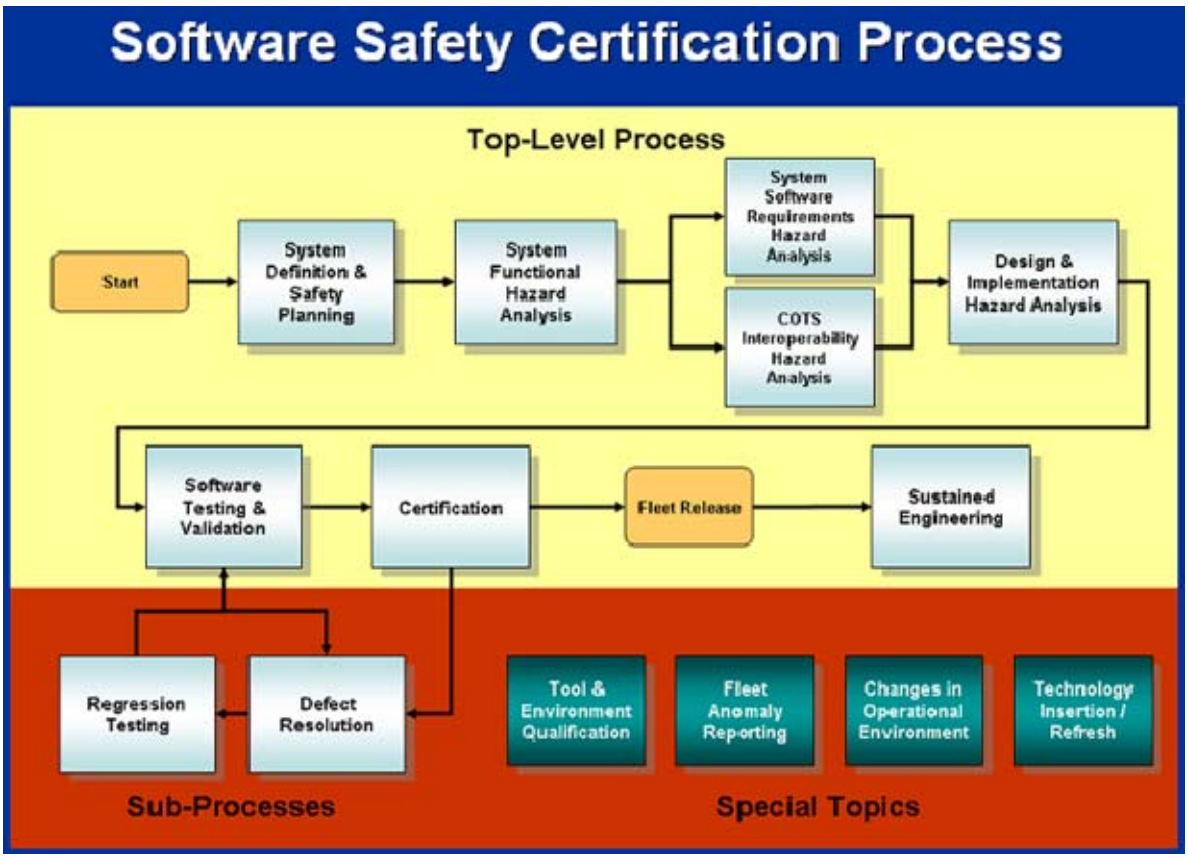


Figure 1: WSESRB Structure (From NAVSEAINST 8020.6D)

Figure 1 represents the WSESRB certification process (Naval Sea Systems Command, 1997). The WSESRB's responsibility is to review safety aspects of each weapon system in order to ensure the Navy's safety requirements are met. The software engineering processes are not directly addressed within this certification process; instead, software engineering processes are handled through the SSSTRP, a subcommittee that addresses software development processes and outputs in order to ensure software safety. The software vendor responds to the SSSTRP's Request for Proposal (RFP) with a predefined Technical Data Package (TDP). The TDP requirements structure lacks a standardized method of evaluation (Rivera & Luqi, 2010).

1. Problem Statement

A gunship system has both hardware and software components. Unacceptable unintended behavior of the software system may result from defective architectural changes made to the hardware and/or software components of the gunship system. The defective architectural changes can result from an incorrect implementation of well-designed software system architectural plans and/or the correct implementation of a software system architectural design that does not meet the gun weapon system requirements. The Navy's Software System Safety Technical Review Panel (SSSTRP), a committee of domain experts, is responsible for evaluating the gun weapon system architectural designs, but its evaluation methodology does not contain adequate structure for evaluating potential gunship architectural changes and/or the software tools necessary to test the proposed gun weapon system architectural changes. Consequently, the SSTRP committee would unwittingly approve of defective software system architectural changes that can result in unacceptable unintended software behavior, which, in turn, can lead to potential software safety risks. These potential software safety issues, if unidentified during the SSSTRP evaluation process, can eventually derail the gunship system acquisition. To identify potential software safety issues that may bring such demise to the gunship system acquisition, it is necessary to achieve these two goals: (1) Identify areas within the SSSTRP evaluation process that need improved and (2) predict the unintended behavior of the gunship software. A research effort is thus needed to enable attainment of the two goals. It consists of an investigation of the SSSTRP evaluation process and the development of a software tool that has the ability to model potential gunship software system architectural change. The investigation of the process will result in recommendations for improving the SSSTRP evaluation process. The software tool will aid the SSSTRP personnel in the evaluation of potential software system changes.

1. SSSTRP Mission

The SSSTRP's primary focus is to investigate the vendor's software engineering processes, and to identify any risks associated with the implementation of the product. Vendors submit Technical Data Packages (TDPs) that contain supporting documentation from the vendor's software engineering quality assurance program. The vendor's responsibility during the SSSTRP presentation is to explain the known risks of its product, and the risk mitigation strategies for each known risk.

The design of the SSSTRP review process entails assignments of both functional and subject matter experts (FME/SME) as members of a technical review board. The TDP is comprised of software development life cycle documentation that was generated during the vendor's product development process.

Our research shows that the current SSSTRP process has a failure rate of over 80% (Rivera & Luqi, 2010), resulting in (1) the government program office placing the project on hold until the vendor responds to the failures; or (2) the government acquisition community having to find an alternative vendor solution that has the functional and technical capability to pass the SSSTRP process.

A vendor's failure in the SSSTRP process may impact both the end-user and the acquisition community in the following ways:

- Project timelines are at risk, thereby resulting in higher failure rates for related project milestones.
- The end-user ability to leverage the new product functionality/capability is delayed.
- The end-user may be forced to use a product that has lesser functionality overlap, or multiple products to meet the total functional requirement.
- Acquisition processing costs may be higher, with lower customer satisfaction.

The unacceptable risks associated with the high level of SSSTRP failures are due to a SSSTRP evaluation process that has no clear definition of software analysis, and no identification of a standardized evaluation process. The purpose of this research is to explore the problems of the naval gun weapon system SSSTRP evaluation process, and propose a methodology for identifying software safety risks. Specifically, our research investigates how to reduce the impact of the vendor failing the SSSTRP process, and how to standardize the software safety quality assurance requirements using a formal method of evaluating potential software.

2. Research Approach

The primary goal of this research project is to identify SSSTRP evaluation process improvements, and provide a methodology and tools that support a software safety assessment with higher fidelity. Figure 2 represents the approach used for this research:

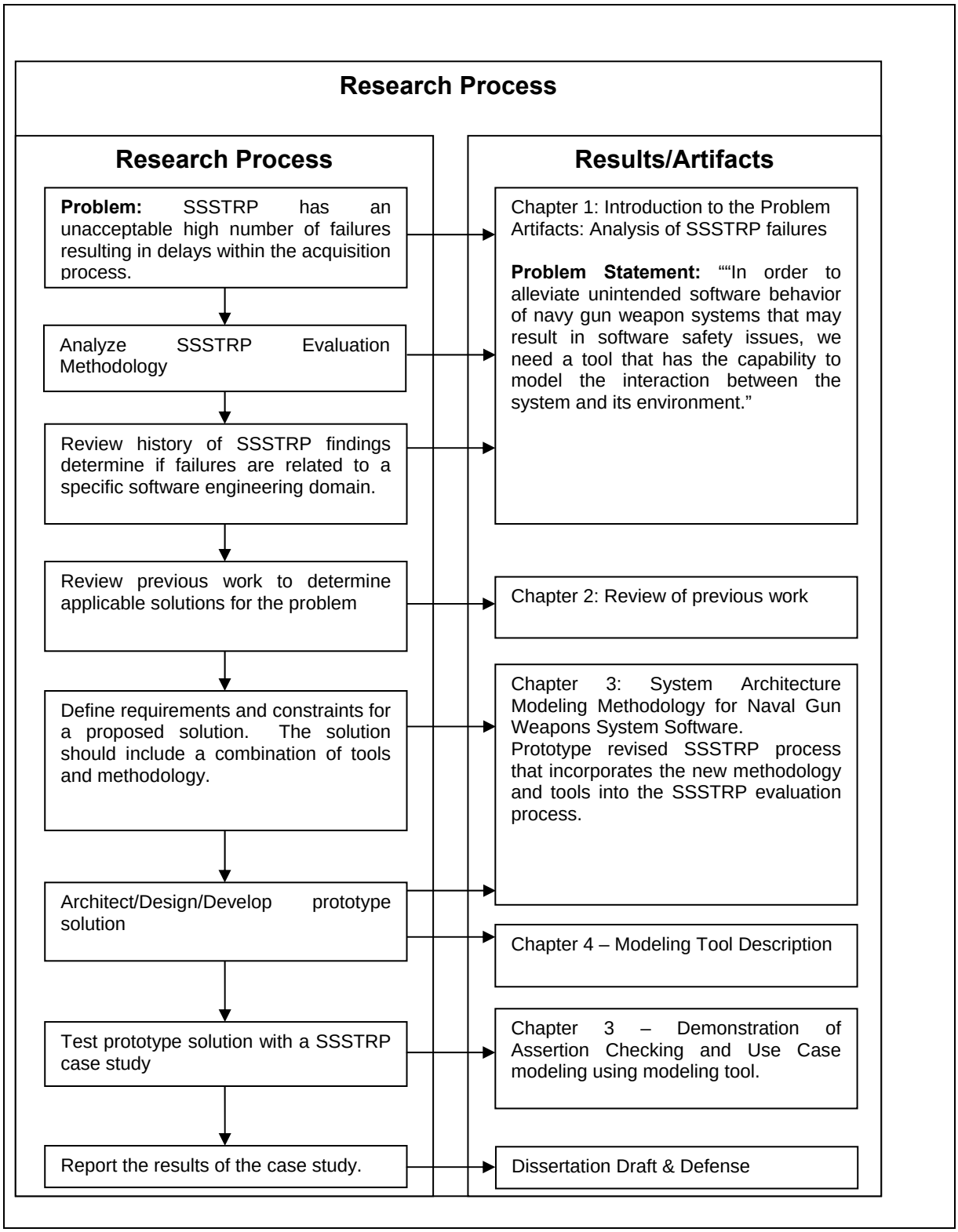


Figure 2: Dissertation Research Process

The research plan includes the following areas of focus:

- Analyze SSSTRP Reports – The purpose of the analysis phase is to identify the primary reasons for SSSTRP failures within a data set that contains 2007-2009 unclassified U.S. Navy SSSTRP reports.
- SSSTRP Structure – The research plan includes a requirement to analyze SSSTRP personnel structure, and the impact of the SSSTRP personnel structure relative to vendor failure rates.
- Identify Modeling Methodologies – Recommendations for improvements to the SSSTRP process may require an integration of a modeling methodology that supports a streamlined acquisition process, and ensures a high-level of fidelity relative to software safety evaluation techniques.

The research goals are used as a means of determining the course of the research. The goals have also been established based on the literature review, which identifies potential gaps in the current Navy software acquisition process.

The research for this dissertation required us to submit a request to the U.S. Navy to view the previous three years of naval gun weapon system SSSTRP findings (2007–2009). The Navy Program Office PEO IWS 3C, Naval Gunnery Project Office approved our request to release SSSTRP results that were not classified as sensitive, and provided a subset of three years of SSSTRP findings. The SSSTRP findings contain opinions, reports, and recommendations to the WSESRB. Issues identified in the SSSTRP are documented in the final report (Rivera & Luqi, Requirements Framework for the Software System Safety Technical Review Panel Technical Review Package, 2010).

The SSSTRP reports were analyzed in order to determine potential commonalities for vendor failures. The SSSTRP failures were categorized using SSSTRP failure category definitions that were obtained from the WESESRB directive NAVSEAINST 8020.6D (Naval Sea Systems Command, 1997):

a. Project Planning

The Software Development Project Plan defines the dates, milestones, and deliverables that drive the project's milestone and timeline definitions. The following documents are software engineering project management deliverables that fall within the "Project Planning" category:

Project Charter – The Project Charter describes the agreement between the organization providing the product or service, and the client organization requesting and receiving the project deliverable. It is a tool to obtain commitment from all affected groups and individuals within a specific project. It is an agreement between the technical and business groups which define:

- Partners and external stakeholders
- The project management framework
- Roles, responsibilities, accountabilities, and activities of the team members
- Management commitments
- The authorized project accountability framework

Project Management Plan (PMP) – The PMP is the controlling document to manage an Information Management/Information Technology (IM/IT) project. Upon approval, the PMP provides a baseline to monitor progress and measure results. The PMP contains the following structure:

- Purpose, scope, and interim and final deliverables of the project
- Schedule and budget for the project
- Project assumptions and constraints
- Managerial and technical processes necessary to develop the project deliverables

- Resource requirements
- Additional project plan requirements

Scope Statement – The Scope Statement is a summary-level description of a project that includes project justification, project purpose and scope, and high-level work plan and deliverables, in addition to product/service description.

Quality Management Plan(QMP) – The QMP describes the requirement to ensure the products/deliverables are correct (i.e., function correctly, satisfy specifications) and to ensure that the project's project management and development processes are applied properly so as to ensure the quality of the products.

The Quality Management Plan identifies the standards, practices, and methods to be used in the project for performing quality assurance activities. It also explains the verification process for deliverables, the tracking and reporting of items that do not conform to the QMP, the process to approve deliverables, and the process for Technical Reviews and Verification and Validation Audits.

Test Plan – The Test Plan is used to organize, schedule, and manage the testing effort. The test plan defines the types of testing (e.g. functional, performance, usability) and the test levels (e.g., unit, integration, field testing) within the planning and implementation phases of the project.

The Test Plan identifies test items, testing tasks and responsibilities, the testing environment, testing resource requirements, and the schedule of the testing activities. It also lists the individual tests, and the objective, procedures, and expected results of each test.

Risk Management Plan – The Risk Management Plan describes the management of project risk, and is a subset or companion element of the Project Management Plan. It identifies the involvement of the project team, the supplier, and the client in executing risk management activities, the detail and

scheduling of each major risk management activity (e.g., identification, analysis, prioritization, monitoring), risks threshold criteria, and reporting formats.

Performance Plan – The Performance Plan specifies the project parameters (e.g., cost, schedule, risks) and the product/service attributes (size, complexity, sites) that will be used to analyze and report the current status of the project, and to forecast future progress and status. It is a subset or companion piece to the Project Management Plan.

The Performance Plan outlines what raw data will be collected, the performance requirements analysis plan, the performance testing tools, and types and frequency of performance reports.

HR (Staffing and Training) Management Plan – The HR Management Plan defines the rotation schedule for project resources, and the evaluation of performance. In addition, it identifies the training requirements to ensure the project team posses the requisite knowledge and skill set.

Configuration Management Plan (CMP) – The CMP describes the set of activities and tools to ensure that the project has adequate control over all items necessary for creating or supporting the project deliverables. The CMP defines the project deliverables in which it has control, and the mechanism for controlling changes to those items. It also describes how baselines are produced, the configuration reporting requirements, and the audits or reviews of the configuration management process.

Procurement Management Plan – The Procurement Management Plan documents the management process of identifying how project needs may best be met by procuring products and/or services, such as:

- Hardware (e.g., development and/or installation hardware)
- Software (e.g., COTS, outsourcing some or all of the development)
- Services (e.g., management or development contractors/consultants)

The Procurement Management Plan identifies procurement strategies, outlines the scope of products and/or services to be procured, and identifies responsibilities for the procurement process up to and including contract closeout.

Requirements Management Plan (RMP) – The RMP describes the management of the project's requirements for products and services during the project life cycle. It describes the steps to develop an understanding of the provider's requirements with specific focus on requirements definition and measurements. The RMP also identifies and controls changes to requirements as they evolve during the project to ensure traceability.

Software Development Plan (SDP) – The SDP details the activities and deliverables during the Software Development Life Cycle of a project. The SDP defines the software development methodology, the design, programming and documentation standards, the establishment, control, and maintenance of the development environment, and any other applicable software development activities.

Information Management Plan – The Information Management Plan details the communication and integration activities required to successfully incorporate the new functionality in the enterprise, to include ensuring the new product is in accordance with existing legislation, regulations, and policies.

The Information Management Plan describes the identification of client information needs, and the information standards. In addition, the Information Management Plan describes how access to information, privacy, confidentiality, security, intellectual property provisions, retention requirements, and other life cycle management of information considerations are taken into account within the project life cycle.

Requirements Specification – The Requirements Specification defines the boundaries for the project and explicitly specifies system/product requirements and features. The Requirements Specification stipulates functional, performance, information, capability, safety, security, ergonomics, operations,

maintenance, interface, qualification requirements, and the definition of acceptance criteria. The Requirements Specifications provides a documented reference of the project team's understanding of the product/system requirements, and the deliverables required to provide the product/system.

Risk Log - The Risk log is a listing by ranking of the project risks and related risk information. The Risk Log provides a statement of each risk, its ranking, the probability of occurrence and impact if the risk occurs, the planned response, the person responsible for mitigation actions, and the current status and actions.

Change Requests - Changes occur during the project life cycle due to the addition or change to the requirements of the project's products or services, to an increase or decrease in the complexity of project activities, to an under or over cost or time estimate, or due to changes in the project assumptions or dependencies. A Change Request identifies the need to expand or contract the project scope, modify costs or adjust schedule estimates. It describes in a concise manner the reason, scope, and impact of a change, and records the approval to proceed with the change.

Closure Plan - A Closure Plan summarizes the results of a project and the activities required for the transition of the project's products and services from "development" to "production" state. The Closure Report identifies the extent to which the project objectives were satisfied and the anticipated benefits realized, the person or group within the client's organization who will oversee the transition to the "production" state, the lessons learned during the project, the list of project files, and the support arrangements and warranty period, rules and conditions.

Project Acceptance Plan – The Project Acceptance Plan formalizes client acceptance of all the deliverables of a project (or a phase) and also confirms that there are no outstanding deliverables.

Deployment and Maintenance Plan – The Deployment and Maintenance Plan is a high-level design of the approach to system maintenance.

This concept sets the overall parameters for change management during the maintenance phase. Version control, upgrade planning, and legacy support are parts of the Deployment and Maintenance Plan.

b. System Safety Program

The System Safety Program optimizes system safety in the design, development, use, and maintenance of software systems and their integration with safety critical hardware systems in an operational environment.

Software Safety Program – The Software Safety Program identifies all potential risks associated with a software installation, usage, interface requirements, hardware/software sharing, software maintenance, and system retirement. The Software Safety Program identifies critical risk scenarios that affect the software's ability to function not as designed, or to mitigate functional design risks.

Safety Risk Management - Safety Risk Management is an iterative process that begins with an initial safety assessment of all known hazards. Known hazard states are stored in a Hazard Tracking System (HTS) in order to document the mitigation associated with each hazard. Safety precepts are incorporated during system development to reduce the likelihood of the hazards from occurring. Safety Risk Management concludes when the residual risks have been reduced to a level acceptable to the appropriate authority.

Safety Verification/Audits - Safety Verification and Audit efforts are performed to ensure safety data is being collected and objectives and requirements of the safety program are being met. Test plans, test procedures, and results of all tests including design verification, operational evaluation, technical data validation and verification are reviewed to ensure the safety of the design is adequately demonstrated and that the results of the safety evaluations are included in the appropriate test and evaluation reports. Audits are scheduled at major program milestones so as to provide management with an indicator of safety program progress.

Hazard Tracking Management – The purpose of Hazard Tracking Management is to identify safety critical issues, evaluate hazards, and document/manage the mitigation efforts required to minimize the impact of the hazard. Tracking systems are part of the risk mitigation strategy and are an ongoing effort to stabilize the safety of control-based software.

Based on the software test data, (including mishap data from similar systems and other lessons learned), hazards associated with the proposed design or function are evaluated for hazard severity, hazard probability, and operational constraints. As a minimum, the Preliminary Hazard Analysis considers the following for identification and evaluation of hazards:

- Hazardous components (e.g., fuels, lasers, toxic substances, munitions).
- Safety design criteria to control safety-critical software commands and responses (e.g., inadvertent command, failure to command, untimely command or responses) must be identified and appropriate action taken to incorporate them into the software specifications.
- Environmental constraints including the operating environments (e.g., temperatures, fire, lightning, and radiation).
- Safety related equipment, safeguards, and possible alternate approaches.
- Identification of the safety requirements, standards and other regulations pertaining to personnel safety, environmental hazards, and toxic substances with which the system will have to comply.

COTS/GOTS/NDI Assessment - In an Open Architecture environment, the COTS/GOTS/NDI assessment is centered on the exposure related to COTS/GOTS/NDI software that shares the same architectural support.

The interfaces between the COTS/GOTS/NDI software and the Open Architecture environment are thoroughly analyzed to ensure there are no impeding conflicts. The ongoing management and/or maintenance of COTS/GOTS/NDI software is also monitored to ensure version control is well-documented and analyzed in order to ensure the vendor has not made changes that may impact the safety assessments.

Simulation/Stimulation/Emulation - Simulation-Stimulation-Emulation test documentation is evaluated to ensure proper stress, boundary, and environmental testing meets the minimum software system safety requirements for Open Architecture integration.

C. RESEARCH FINDINGS: SSSTRP REPORT ANALYSIS

The process of reviewing three years of SSSTRP reports was designed to identify potential gaps in the current software acquisition SSSTRP evaluation process, and to identify trends in TRP vendor failures. A failure is defined as any SSSTRP report that resulted in the software acquisition process being temporarily or permanently halted as a result of the SSSTRP review. This section covers the methods used to gather, classify, and report the SSSTRP failures.

A total of 86 SSSTRP reports were identified within the 2007, 2008, and 2009 fiscal years. Table 1 provides a summary of the resulting issues within these reports. It should be noted that although the total number of issue reports was 86, there are 177 total issues reported within these issue reports; this is due to multiple SSSTRP reports containing multiple failures within multiple categories. With a mean of 2.06 failures per SSSTRP failure report, the maximum number of failures found was four and the lowest number of failures being one. SSSTRP failure reports are not issued in cases where there is no failure found.

Category	# Issues	Percentage of all Failures	Percentage of SSSTRP Reports
Software Safety Program	34	19.2	39.5
System Safety Program	27	15.3	31.5
Safety Verification/Audits	24	13.6	27.9
Product Integration and Test	19	10.7	22.1
Project Planning	18	10.2	21.0
Safety Risk Management	12	6.8	14.0
Validation & Verification	9	5.1	10.5
Risk Management	9	5.1	10.5
Configuration Management	6	3.4	7.0
COTS/GOTS/NDI	6	3.4	7.0
Hazard Tracking Management	5	2.3	5.8
Sim-Stim-Emulation	4	2.3	4.7
Requirements Management	2	1.1	2.3
Deployment & Maintenance	2	1.1	2.3

Table 1: Ungrouped SSSTRP Failure Results

The figures in Table1 indicate ungrouped failures for all potential failure modes identified within the 86 cases that were analyzed. The results show that the majority of SSSTRP failures were found within the more complex areas of project management and system and software safety management. Table 2 indicates the number of failures found within these areas (as well as those that belong to other stages, such as maintenance/implementation).

Stage (grouped Categories)	# Issues	Percentage of all Failures
Software and System Safety and Risk Management	111	62.7
Project Management (Implementation)	58	32.8
Life Cycle (Post implementation)	8	4.5
Total	**177**	

Table 2: Grouped Error Reports in SSSTRP Failure Reports

Table 2 shows that approximately 63% of all reports occurred within the system and software safety and risk management areas, which include Software Safety Program, System Safety Program, Safety Verification/Audits, Safety Risk Management, Risk Management, and Hazard Tracking Management. The second most common area of problems reported was in the Project Management/Implementation category, which included errors in the Product Integration and Test, Project Planning, Validation and Verification, COTS/GOTS/NDI, Simulation-Stimulation-Emulation, and Requirements Management phases of the vendor implementation plans. Approximately 5% of final failures occurred post-implementation in the project life cycle; these failures fell within the Configuration Management, and Deployment and Maintenance phases.

The conclusive evidence within the research shows the majority of weaknesses within the SSSTRP process for Navy software acquisition occurs within the system and software safety areas. These findings are also consistent with previous research in the area, which found that safe software acquisition was increasingly complex and was a consistently problematic area in Naval acquisition processes (Rivera & Luqi, 2010).

D. LIMITATIONS OF STUDY

The SSSTRP reports that were made available by the U.S. Navy included only declassified results. The representative nature of these reports was impossible to determine with certainty because we did not have access to the full set of the data, which included classified SSSTRP reports. However, because the structure of the SSSTRP evaluation process is not affected by the classification level of the data, there is no reason to believe that the unclassified data used for this research is not representative of the domain.

As the reports were provided over a period of three years, it was expected that there would be variations in format and textual content. However, the structure of the reports varied and the reports did not display a consistent methodology for reporting SSSTRP findings. Although the reports did have

general headings that could be used for guidance, the data contained in the reports was opinionated justification for recommendations, and did not follow a standardized evaluation and reporting format. Over the three years of report data, there were few instructions for how a SSSTRP member should report a finding, and what information was required for a failure report.

This lack of evaluation and reporting structure resulted in SSSTRP evaluation reports that were largely the result of human inspection, which led to a SSSTRP member's personal opinions about a potential failure. Thus, the inconsistency and weakness in internal structure of these reports made them impossible to categorize/analyze beyond the classifications found in this dissertation. Additionally, a large number of issues resulting in a SSSTRP failure were present in one or more milestone phases. Since the evaluation milestones are sequential, this was highly problematic because issues that were found in earlier stages were problematic for future milestone requirements. "Repeating Incidents" was widespread and persistent, and was identified across multiple project milestones.

Related to the SSSTRP evaluation process is a human resources issue that was identified during the examination of the reports. Our research found a very high turnover on the SSSTRP committee, with few long-term members. Additionally, the SSSTRP committee did not always consist of software or process experts, but included members from other areas of expertise. The committee members, in addition to rotating frequently, also did not have standardized evaluation documents available in order to ease the process of failure determination; instead, each failure was identified, analyzed, and processed individually.

1. Vendor Self-Assessment

The SSSTRP reports did not demonstrate any evidence of self- of the software systems submitted by vendors. The research showed a lack of readily available standards for this self-assessment, preventing software vendors from routinely determining whether their products would meet the demands of the

SSSTRP. Requirements documents were available that identified the functional requirements of the software; however, these documents did not identify the safety requirements and risk assessment processes used for the software. Additionally, as the SSSTRP committee does not have a guideline for the analysis of the safety requirements or other requirements of the submissions, it is difficult for vendors to determine what will (or will not) pass the screening process. Because of this ambiguous evaluation process, it is exceptionally difficult for a vendor to determine potential areas of TRP improvement that could increase the vendor's chances of passing the SSSTRP evaluation process. By extending the TRP to include a vendor self-assessment, it would be possible to improve the overall outcomes of the process and increase the chances for the products to pass the SSSTRP assessment.

2. Research Results Summary

The analysis of the SSSTRP reports resulted in the following summary:

- The SSSTRP is unable to sufficiently test potential naval gun weapon system software solutions during the acquisition process.
- The Vendor Technical Data Package (TDP) requirements and evaluation methodology is not structured in such a way that supports a high fidelity evaluation of software safety.

The main recommendation that may be derived from this analysis is that the SSSTRP review process may be improved with the introduction of a methodology that can be used by both the SSSTRP members and software vendors to evaluate software safety. Providing the SSSTRP community with high-level models that may satisfy a portion of the software safety assessment process improves the current inspection-based evaluation methodology. Without a high-level modeling process, the alternative is to implement the system and to perform testing. Manual testing is a very expensive and timely alternative, which may be partially satisfied using the prototype methodology and tools that are covered in Chapter III. Additionally, consider the following advantages:

- Short Feedback Cycles - Automating system/business processes start with process design. The creative process of redesign requires iterations of your ideas. The modeling timeline should be as short as possible to align the results of each step with its input. MP satisfies this requirement in short order as there are only seven total constructs required to model in MP.
- Involving Domain Experts in Model Development - Because MP is easy to learn, teaching domain experts how to model in MP is critical to shortening the SSSTRP evaluation process. Closing the knowledge transfer gap between business and IT may result in models that require less testing and include lower levels of refinement. Our modeling methodology suggests domain experts are part of the model development team. As identified in my dissertation (Prototype SSSTRP Evaluation Process), the domain experts do not create all models on their own, but they are a part of the team with technical people.

Chapter II describes the research for suitable software safety modeling frameworks.

II. REVIEW OF PREVIOUS WORK

A. INTRODUCTION

The literature review discusses the current state of methodology and tools within software safety domain, with specific focus on enterprise systems. This review contains information about the COTS integration risks, vendor selection, software acquisition, software architecture, the Navy's Open Architecture Enterprise Program, abstract modeling methodologies, and software safety standards and frameworks. The goal of this chapter is to review the current state of technology in order to determine if a potential solution exists that may reduce the vendor failure rates within the SSSTRP process.

The Navy's SSSTRP process has been using both commercial off-the-shelf systems (COTS) and open architecture (OA) approaches to satisfy the software requirements for new and emerging technologies associated with naval weaponry. The acquisition process of COTS-based software exposes an organization to the potential for operation failure due to discontinued support of the product; acquisition or dissolution of the vendor; or aging software becoming less compatible with newer software that has related functions. However, the risk profiles of (COTS) software and customized software vary and may provide different advantages and disadvantages to the implementation of new systems. In an attempt to evaluate the effects of both COTS and OA approaches to naval weaponry software safety requirements, the literature review chapter covers both COTS-based solutions and custom development.

B. SOFTWARE SAFETY RISKS WHEN EVALUATING A COTS SOLUTION

COTS software is a popular software choice for organizations that want to acquire and implement software quickly and easily. However, lack of control over the software configuration or lack of ability to customize the software may lead to a less than optimal solution for the software installation, as well as making

the customer unduly dependent on the vendor. Chapter II.B. discusses the software safety risk assessment process for COTS, as well as the particular risks associated with COTS software acquisition.

C. GUN WEAPON SYSTEM SOFTWARE SAFETY RISK: SOFTWARE OBSOLESCENCE

One of the major software safety risks of COTS is obsolescence (Merola, 2006). Merola defined software obsolescence as follows: "Software applications become obsolete when they are retired from use and taken off the market due to technology advancements, decrease in product popularity, or other market factors." Merola studied the issue of software obsolescence in military applications, where systems development moves slowly enough that software is often considered to be obsolete in civilian systems before it even makes it into military systems. While most civilian development does not move as slowly as military development, the problem also plagues the civilian market. Merola described software obsolescence risk in the systems design, integration, production, and program management environments rather than in the operational environment. Merola remarked that the specific risk in software obsolescence was, "the inability to maintain an infrastructure to properly integrate the systems, develop, maintain, or troubleshoot hardware or software code." Merola distinguished between logistical, technical, and functional obsolescence as well. Logistical obsolescence is the point at which system support, new licenses, or expansions are no longer available from the vendor (Merola, 2006). Functional obsolescence occurs when the software no longer functions as required or cannot be modified to perform required tasks, and technical obsolescence occurs when technical specifications of the system have been overtaken by technical advances (Merola, 2006). Thus, a system may remain functionally useful even after it has become technically obsolete, and it may remain technically and functionally useful even after its logistical obsolescence.

Merola provided a number of recommendations for avoiding software obsolescence risk when utilizing COTS products for systems development. The

first of these recommendations was to include an analysis of potential obsolescence in the market analysis research typically performed during the software acquisition process (Merola, 2006). The market analysis is performed to determine relative software quality, cost, and other features of the software. Although software vendors are not typically willing to reveal their software obsolescence plans, an examination of the vendor's historical patterns of software obsolescence and their software renewal cycle may provide insight into the likelihood of obsolescence in the product being chosen (Merola, 2006). The market analysis should include not only the vendor's obsolescence planning, but also an investigation of the current state of the technologies in use and how they may engender technical obsolescence in the near future (Merola, 2006). Although the market analysis cannot prevent all potential risk from software obsolescence, it may help an organization avoid implementation of a system with potentially obsolete COTS components, as well as giving a potential timeframe for the obsolescence of the component in use (Merola, 2006).

Leveson argued that the key to understanding safety lies in the understanding that no one component failure or no human error ever occurs in isolation - an accident is a result of some systemic problem (Leveson, 1995). Leveson argues that more than ever, software engineers/architects/managers must understand the responsibilities of software safety and develop the skills needed to anticipate and prevent accidents before they occur. Professionals should not require a catastrophe to happen before taking action. Leveson examines the following software safety fundamentals:

- Demonstrate the importance of integrating software safety efforts with system safety engineering
- Describe models of accidents and human error that underlie particular approaches to safety problems
- Present the elements of a software program, including management, hazard analysis, requirements analysis, design for safety, design of the human-machine interface, and verification

Software allows unprecedented levels of complexity and new failure modes that are starting to overwhelm the standard approaches to ensuring safety. Software obsolescence in naval gun weapon systems carries an inherent risk that requires continual software safety assessments, with specific focus on how the software acquisition process may affect legacy systems.

D. VENDOR SELECTION SOFTWARE SAFETY RISKS

The choice of vendor for the provision of gun weapon system changes carries with it a number of risks that may affect the SSSTRP evaluation process. These risks include product availability, schedule slippage, vaporware, modifications to the product that disrupt system compatibility or design, inter-component compatibility, and lack of continued support (Rehman, Yang, Dong, & Ghafoor, 2005). As in obsolescence and requirement mismatches, a market survey before the choice of a COTS vendor will mitigate the potential for vendor-based risk, but there is no way to eliminate the risk due to changes in software components, something that all software undergoes as new vulnerabilities and bugs are exposed (Rehman, Yang, Dong, & Ghafoor, 2005). Therefore, a decision-making framework is needed to help minimize the risk from vendor changes and unreliability that may help in avoiding some of the more common risks, such as schedule slippage and vaporware (Rehman, Yang, Dong, & Ghafoor, 2005).

E. REQUIREMENTS AND COTS CAPABILITY MISMATCHES: A SOFTWARE SAFETY RISK

A potential SSSTRP evaluation risk is the mismatch between systems requirements and COTS capabilities. "Such mismatches are inevitable as COTS products are made for broad use while system requirements are specific to their context," (Mohamed, Ruhe, & Eberlein, 2007). The issue of COTS capability mismatch is particularly relevant to naval gun weapon systems as a significant risk may be posed for successful product integration into an existing system. If the degree of mismatch is too great between the system requirements and the capability of the chosen COTS system, excessive resolution costs for providing

"glueware" and other customized changes to the system may result in unidentified software safety risks, or the system could simply be made unsuitable for the use to which it will be put (Mohamed, Ruhe, & Eberlein, 2007).

The use of formal decision support systems, rather than an ad hoc approach has been recommended (Mohamed, Ruhe, & Eberlein, 2007) to resolve these requirements/capabilities mismatches. Also recommended is the use of a formal method to determine if the mismatches can be resolved and, if so, the most efficient choice of resolution methods (Mohamed, Ruhe, & Eberlein, 2007). A decision support framework, called Mismatch Handling for COTS Selection (MiHOS), is provided as a means of comparing the cost, effort, and risk of resolution actions for requirements/capabilities mismatches in COTS-based software implementations (Mohamed, Ruhe, & Eberlein, 2007).

F. SOFTWARE ACQUISITION EVALUATION: PERFORMANCE AND RELIABILITY

In a naval gun weapon system, software performance and reliability are high-priority requirements when evaluating a potential software solution. The development of nonfunctional requirements, including performance aspects, software quality, speed of execution, and other quality of service factors was a latecomer to the component-based architecture paradigm (Bertolino & Mirandola, 2004). Initially, component-based architecture was concerned only with functional specifications—the way in which the component could be used and the component's functional purpose. In order to ensure performance of component-based systems, care must be taken in the architectural specification and development of the system, and as much information as possible about the components must be gained (Bertolino & Mirandola, 2004). However, this approach may not be sufficient because component-based architecture is often relying on different systems that may not be fully compatible with each other. This may slow development or reduce the performance of the system.

Reliability is another software safety consideration that may affect the overall usefulness of the component-based system. "One of the motivations for

specifying software architectures explicitly is the use of high level structural design information for improved control and prediction of software system quality attributes" (Reussner, Schmidt, & Poernomo, 2003). However, with component-based architecture, in some cases the specifics of the individual components (particularly nonfunctional characteristics such as component reliability) may not be known explicitly and must be determined either during the implementation of the system or, less ideally, after the system has entered use. Some specific attributes that may not be known about the component include the usage profile and the required context (Reussner, Schmidt, & Poernomo, 2003). The usage profile of a software component includes how often it is used as well as under what circumstances; thus, a software component that is used infrequently may not be as well understood as those that are used frequently. The required context includes the other components within the system, as well as external components like middleware, operating systems, and network services, any of which may prove to be unreliable.

These parameters were used to create a predictive model of software reliability that took into account not only the component reliability, but also the potential interface with external components (Reussner, Schmidt, & Poernomo, 2003).

Tamura et al. provided a similar model which uses a stochastic approach to software reliability (Tamura, Yamada, & Kimura, 2006). These authors specifically recommend including the integration and testing stages of development in the main software development phase, in order to head off any difficulties observed with the component's reliability during the design stage (Tamura, Yamada, & Kimura, 2006). Although these models are highly technical, they can be used by component-based system architects to detect issues with component reliability and circumvent them by either redesigning the system dependencies or choosing a more reliable component. A more user-friendly modeling methodology that allows the nontechnical stakeholder to visualize the potential software safety scenario is needed. Additionally, integration of the models by Tamura et al. and Reussner et al. are not usable by SSSTRP

evaluation process due to their level of complexity, the time requirements for development, and the inability of the stakeholder to understand the results.

G. SOFTWARE ARCHITECTURE MODELS AND CONSTRAINTS

UML (Unified Modeling Language) is commonly used to design and analyze component-based systems (Booch, Jacobson, & Rumbaugh, 2000) (Coronato, d'Acierno, & De Pietro, 2005), and is the current tool of choice when modeling naval gun weapon systems. Specifically, D.Harel's state charts are commonly used when modeling system states (Booch, Jacobson, & Rumbaugh, 2000). The major problem with state charts is that the process of creating a formal relationship between the system and the state chart is extremely difficult and highly complex, and thereby too time consuming to be practical for the navy gun weapon system acquisition process.

These modeling practices are meant not only to provide a blueprint for the system design, but also to test the system's fidelity to requirements and design specifications following implementation. Coronato et al. (2005) remarked that:

> By defining the fidelity of the model as the measure of the correspondence between the model and the final system, it can be stated that UML enables designers to produce low fidelity models to capture high-level system characteristics in the early design phase, as well as high-fidelity models to specify low-level system details in the late design phase. (Coronato, d'Acierno, & De Pietro, 2005)

Other significant advantages of using the UML modeling specification are that it creates a standard representation for use between development teams, such as development efforts between a potential vendor and the organization contracting the development. One problem with the use of UML, however, is that there is no way to represent design constraints upon the system, particularly during translation to another language for implementations, such as IDL (Implementation Definition Language) (Auguston, Program behavior model based on event grammar and its application for debugging automation, 1995). According to these authors, "Design by Contract" is the practice of contracting a vendor or outsourcer to provide custom or semi-custom software. The "Design

by Contract” practice is dependent on the availability of constraints in order to enforce the design practices required within the system (Auguston, Program behavior model based on event grammar and its application for debugging automation, 1995).

Constraints, which are derived from high-level business requirements or business rules, provide the explicit requirements definition for the software system design. Constraints can be difficult to manage between software components, especially in cases where the components do not have a consistent way in which they handle the constraint processing. A modeling language that would allow for the definition of constraints in a way in which they can be passed from component to component is described as Constraint Description Language (CDL) (Coronato, d'Acierno, & De Pietro, 2005). The CDL language was derived from standard OCL and was adapted to component architecture. It is noted that ignoring the problem of constraints was not possible if the end result of the effort was to be a coherent software system; however, there was no readily available way in which to transfer constraints between different components (Auguston, Program behavior model based on event grammar and its application for debugging automation, 1995). The lack of consistent treatment of constraints between components represents a significant weakness in component-based architecture. As the authors noted, the issue of managing shared constraints is not a difficulty that cannot be overcome; however, it should remain a consideration in development of a component-based software system (Auguston, Program behavior model based on event grammar and its application for debugging automation, 1995).

The development of languages specific to component-based systems architecture and design has been heavily researched over the last ten years, with new architectural languages, such as AAL, being the byproduct (Booch, Jacobson, & Rumbaugh, 2000). Component-oriented programming that implements the systems is a recent development in software engineering. Fabresse et al. described a conceptual language for component-based architecture and design (Fabresse, Dony, & Huchard, 2008). Their language,

SCL (Simple Content Language), described only the basic and essential elements of a component-based design language, as derived from a large number of existing component-based programming languages, like ComponentJ, ArchJava, Julia/Fractal, Lagoona, and Piccola (Fabresse, Dony, & Huchard, 2008). It is noted that the impetus for component-based design has recently shifted from software reuse at design to reduction of evolution costs by design for software reuse (Fabresse, Dony, & Huchard, 2008); thus, it is necessary to have customized ways in which to provide the integration or "glue" that allows components to be combined into a cohesive system.

Combining systems via "glue" has an inherent requirement to evaluate a potential addition to an existing architecture. Auguston (2009) suggests an approach to formal software system architecture specification based on behavior models, (Auguston, Software architecture built from behavior models, 2009). Monterey Phoenix (MP) (Auguston, Monterey Phoenix, or How to Make Software Executable, 2009) is a methodology that defines the relationship between system interaction and the environment. The MP methodology includes the use of event grammar that generates event traces using ordered logic. The MP framework provides the ability to formally evaluate software architecture using assertions. Auguston showed how MP contains the ability to check Assertions. MP is particularly applicable to the naval gun weapon system software safety domain because it (1) is easily understandable by the nontechnical user; (2) supports reuse as the models are designed at the abstract level with no requirement to provide software details; (3) formalizes the evaluation of potential naval gun weapon system software solutions by creating assertions of unsafe software safety states and testing for counter examples of assertions; and (4) can output visual representations of scenarios in formats that are easily understood (Auguston, Michael, & Shing, Environment behavior models for automation of testing and assessment of system safety, 2006).

Auguston's work in Environmental Behavior Models is particularly applicable to the naval gun weapon system software safety domain as the SSSTRP requires exhaustive testing before modifications to a naval gun weapon

system are approved. Jackson's "Small Scope Hypothesis" (Jackson, Software abstractions: logic, language, and analysis, 2006) (Jackson & Damon, Elements of style: Analyzing a software design feature with a counter example detector, 1996) argues that a high proportion of bugs can be found by testing the system within some small scope. Jackson's hypothesis, combined with Auguston's work in environmental modeling, is particularly applicable when attempting to solve the issue of evaluating software safety issues during the naval gun weapon system acquisition process.

Software safety research in real-time systems has led to the development of the Tempo Toolkit. The Tempo Toolkit is an extension of the IOA toolkit, which provides a specification simulator, a code generator, and both model-checking and theorem-proving support for analyzing specifications. The toolkit consists of the Tempo language, which closely matches the format of the pseudo-code used for IOA. The Timed I/O Automaton Language (TIOA) provides the semantic basis for the Tempo Toolset (Archer, Lim, Mitra, Lynch, & Umeno, 2008).

H. SOFTWARE ARCHITECTURE FLEXIBILITY: AN ACQUISITION RISK

The SSSTRP evaluation process was meant to support a streamlined, thorough evaluation of proposed gun weapon system changes. Naval gun weapon systems require architectural flexibility in order to respond to new and improved software capabilities that strengthen a ship's weapons systems. One of the major benefits of component-based architecture is the flexibility that is allowed by the process. Flexibility is necessary because "software needs to be flexible in order to be adapted to new or changing work situations in its context of use" (Wulf, Pipek, & Won, 2008). The flexibility with which software systems are developed will carry through to the implementation stage of the process and will be required to continue past the point of implementation in order to provide for changing requirements. Component–based architecture is ideal for providing flexibility because individual components can be upgraded or replaced as needs change. For example, a system with a user interface component that is separate

from a database component can have its user interface changed as user requirements or technologies evolve, without affecting the existing database component (Wulf, Pipek, & Won, 2008). Component-based development has the potential to reduce maintenance costs, as the components can be updated only as needed, rather than requiring a full refactoring of the system in order to update one part of the system. Wulf et al. described an end-user framework that described a way in which the software development process can be flexible enough to allow changing user needs while reducing the difficulty and maintenance costs associated with these changes (Wulf, Pipek, & Won, 2008).

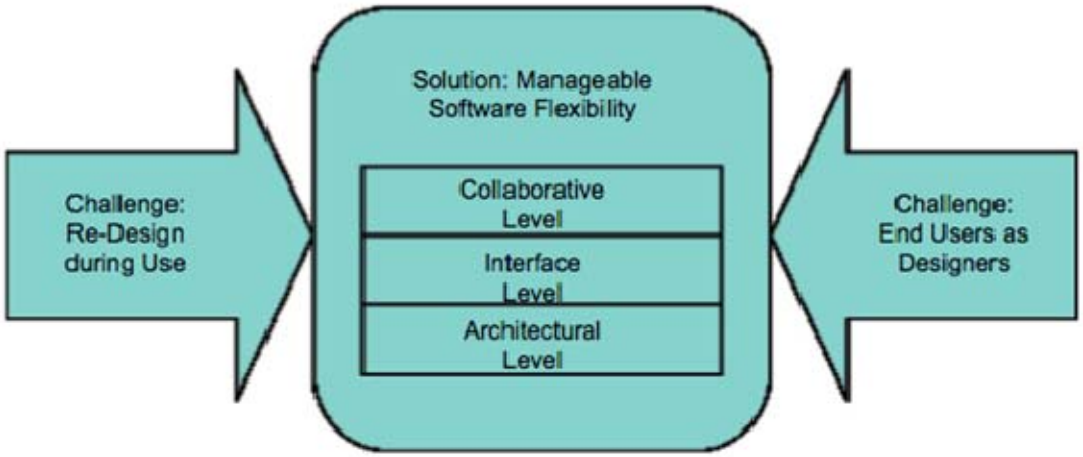


Figure 3: CBD Flexibility Framework (From Wulf, Pipek, & Won, 2008)

Component-based architecture and software design provides a way to design systems to account for system requirements without excessive cost or development time. It is based in assembling software components, which may be either custom-designed vendor-sourced custom components, semi-custom components (such as ERP modules), or commercial off-the-shelf (COTS) software components. Although the component-based architecture process is flexible and modular, there are difficulties relating to the black-box nature of many components, including difficulty evaluating nonfunctional requirements like reliability and software quality, and interoperability between components.

I. DEPT OF THE NAVY OPEN ARCHITECTURE ENTERPRISE (OA ENTERPRISE) PROGRAM

The requirement for naval gun weapon systems to use open architecture was established in 2005 in a memo from the Navy Program Management Office. The memo states that, “Naval OA transformation must match the rapid evolution in commercial and military technology. Not only must we shorten the kill chain across the family of systems; we must also shorten the cost it takes to deliver capability requirements” (Department of the Navy, 2005). Motivations for the adoption of open architecture included reduction of cost and time invested in developing and implementing new systems, and the ability to design systems that are technologically advanced, as compared to the previous development life cycle, in which the end product was typically obsolete by the time it was placed in service (Department of the Navy, 2005). Principles for the OA system implemented by the Navy include the following:

- Modular design and design disclosure
- Reusable software components selected using a best-in-breed strategy, rather than the previous single-vendor strategy
- Interoperable joint warfare communication and information exchange capability
- Design for life cycle affordability, including tactics such as system design and development and support for COTS obsolescence
- Encouragement of alternate solutions and sources in order to improve competitive practices and system capabilities (Department of the Navy 2)

The OA Enterprise system is required to be integrated into all Navy systems and system requirements, and is one of the first identifiable federal programs that require open architecture in the system (Department of the Navy, 2005). The Navy established an Open Architecture Enterprise Team (OAET) to oversee the efforts and ensure that the open architecture requirement was

respected in all ongoing and future Navy system designs. The document also included short-term objectives and system requirements to begin using the OA Enterprise program immediately while the long-term details were worked out. The program has been active since that time.

The Navy OA model is described in the OA Assessment Model (Department of the Navy, 2005). The OAAM is built on a matrix framework using business and architectural/technical characteristics; the level of compliance of each system is assessed on the individual criteria. Figure 3 demonstrates the OAAM's matrix; the chart details the level of business and technical compliance (Department of the Navy, 2005). Each level of the model is accompanied by business integration and architectural technical characteristics; in both axes, "0" represents the least integration of open architecture principles, while "4" represents the highest level.

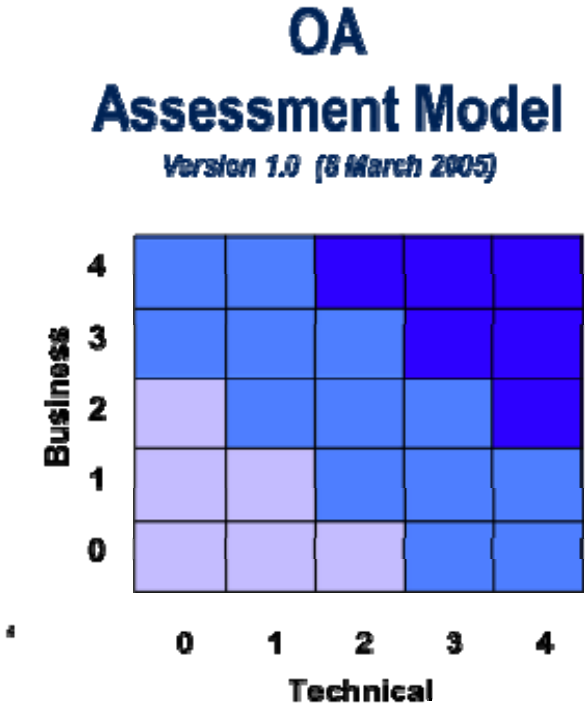


Figure 4: OA Assessment Model Matrix (From Department of the Navy, 2005)

Level	Business	Technical
0	Isolated	Closed
1	Connected	Layered
2	Migrating to openness	Layered and open
3	Common	Common
4	Open and net-centric	Enterprise

Table 3: OAAM Development Levels (From Department of the Navy, 2005)

The OA Enterprise Contract Guidebook is produced by the Navy to ease the integration of open architecture as a design requirement. The OA Enterprise Contract Guidebook offers insight into the program's intentions toward the use of open architecture and how it handles the acquisition of it. The guidebook, designed as part of the Navy's Open Architecture Enterprise (OA Enterprise) initiative, is intended to "provide Program Managers, Contracting Officers, and their supporting organizations with guidance and example contract language to assist them in incorporating open architecture principles into their contracts" (Department of the Navy, 2008). The document also provides insight into the use of open architecture within the Navy, including its history, requirements, and scope.

The intent of the document is not to enforce the use of the language required, but to suggest appropriate language for the contracts used for acquisition of open architecture products. The document also provides an overview of Naval OA architecture and intent. The principles of design include use of both COTS and open standards in order to ensure interoperability and fast-swap capabilities for software, and includes standard interfaces to ensure system communications capabilities (Department of the Navy, 2008). It is noted that regardless of the source of the software component, it should be compliant with the OA Assessment Model (OAAM) at the highest level possible for the given system.

The Contract Guidebook provides insight into the software and systems development process required by the Navy. The OA Enterprise program was undertaken to ensure that the Navy had access to information technology that

was up to date, maintainable, and reliable. By using the open architecture paradigm as a requirement for new systems design, the Navy gained the ability to update its systems easily, to interface its systems, and to ensure that its systems could remain functional in spite of COTS obsolescence. It also placed the government in a stronger position by requiring that the purchasing organization seek out and exercise intellectual property and data rights. Few Navy documents described live projects that had been undertaken using the new guidelines; examples of the outcomes of these guidelines were derived from the literature rather than Navy documentation.

J. SOFTWARE ACQUISITION CHALLENGES OF A NAVAL GUN WEAPON SYSTEM

Testing and software evaluation of a naval gun weapon system that is composed of COTS products is a known problem (Bhansali, 2005). Azani discussed the specifics of testing and evaluation of the open system in terms of strategic requirements and goals (Azani, 2001). Azani noted that the use of open systems provided government IT systems with advantages, including the ability to take advantage of best-in-breed commercial systems and ensure interoperability, commonality, portability within the system, and the ability to replace obsolete systems. Without careful system design, the testing and evaluation of a system assembled from multiple commercial components could be exceptionally difficult to complete successfully.

The design of a testing system that can handle multiple products from various vendors is complex, particularly in cases where some parts of the system may be COTS that do not have open-code bases to allow specific design of the test systems. The testing and evaluation of an open system should be determined before implementing the system, and priority should be given to designing for test and evaluation ease (Azani, 2001). Rajsuman and Noriyuki presented one solution to the problem. The Open Architecture Test System was designed to provide a method to test the implementation and integration of open architecture systems incorporating modules from many vendors (Rajsuman &

Noriyuki, 2004). The architecture proposed was intended to test the full operation of the system. The architecture also allowed for live testing and simulation, and was intended to decrease testing time and simplify the testing process (Rajsuman & Noriyuki, 2004). Integrated system and user test architecture would be a useful addition to an organization transitioning to an open architecture requirements paradigm.

Another software safety issue that may emerge in the use of an open architecture is the dependability of the system. Barrett offered one solution to ensuring reliability in open architecture systems, the Delta-4 project, which is defined as “an open, fault-tolerant, distributed computing architecture for use in application areas such as computer-integrated manufacturing, process control, and office automation” (Barrett, 1993). The system was intended to address the issue of reliability in open architecture systems that were used in applications that required reliable throughput and response time; however, the author noted that the system was not designed for mission-critical or safety applications (Barrett, 1993). The system was based on a Dependable Communication System with the components of the architecture spread through computers and linked by the Dependable Communication System. Software components could be replicated to provide redundancy, with the caveat that host machine configurations had to be consistent across machines in order for the redundancy capability to be used (Barrett, 1993). The communications system allowed for multi-point communication, providing for robust and dependable communication between replicated units. The system also offered fault-tolerance in order to provide a level of protection against hardware failures and a variety of communication mechanisms (Barrett, 1993).

Although Barrett’s system is not intended for mission-critical systems, it provides a blueprint for how the requirement for dependability may alter the design of an open system. Enhancements to the system would be required in order to allow for the level of dependability required in more mission-critical applications, but the system provides a framework for designing a dependable open system.

A third issue in analyzing potential software solutions for a naval gun weapon system is the problem of trusted computing. Naval gun weapon systems have a strict requirement to restrict access to trusted users (and systems) and to assure that security level. Trusted computing within an SoS becomes more difficult because components and their authentication methods may be changed in an ad hoc manner and the overall design of the system may not be set at the initial use of the system (England, Lampson, Manferdelli, Peinado, & Willman, 2003).

K. SOFTWARE SAFETY REQUIREMENTS FRAMEWORKS

Initial searches found numerous frameworks related to software safety. This review is focused on frameworks that make the software package a primary target of the evaluation. While some of these frameworks have been established in the working software development environment, others have only been described within the academic computer science area. The majority of those identified standards are from military or other safety-critical areas rather than from the business or consumer software environment. Most of these standards have been developed for use in military, transportation, medical, communication, and nuclear power systems (Medikonda & Panchumarthy, 2009). As Barrett Medikonda and Panchumarthy noted, most of these systems are real-time control systems, lending an extra level of complexity to safety requirements design.

The research describes a number of frameworks and identifies potential advantages and disadvantages for use within the Naval Weapons Gunfire software systems. Table 4 contains the known software safety requirements standards that use software safety and security features as a main component within the software specification process.

Standard	Description
MoD 00-55	Requirements for Safety-Related Software in [UK] Defense Equipment
MoD 00-56	Safety Management Requirements for [UK] Defense Systems
DO-178B	Software Considerations in Airborne Systems and Equipment Certification
ARP 5754	Safety Assessment Process on Civil Airborne Systems and Equipment
Mil-Std-882	System Safety Program Requirements
Software Safety Hdbk	Software System Safety Handbook
IEC 61508-3	Functional Safety of Safety-Related Systems, Part 3: Software Requirements
IEC 60880	Software for Computers in Safety Systems of Nuclear Power Stations
ANSI/ISA-S84.01	Application of Safety Instrumented Systems for the Process Control Industries
ANSI/AAMI SW58:2001	Medical Device Software Life Cycle Processes
NASA-STD-8719.13	Software Safety
UL 1998	Standard for Software in Programmable Components
EN 50128	Software for Railway Control and Protection Systems
MISRA Auto Std	Development Guidelines for Vehicle Based Software
IEEE 1228	Standard for Software Safety Plans

Table 4: Known Software Safety Standards (Bhansali, 2005)

As noted in Table 4, most of these systems are designed for use in safety-critical real-time applications, indicating that characteristics of any of them could be considered appropriate when examining the potential applicability to the Naval Weapons Gunfire system. However, standards such as ANSI/AAMI SW58:2001, which focus on safety-critical application of medical software, may not be as appropriately applied to the current problem as other defense standards may be. Identified standards that may be most applicable to the current research problem include MoD 00-55, MoD 00-56, DO-178B, Mil-Std-882, Software Safety Hdbk, IEC 61508-3, NASA-STD-8719.13, and IEEE 1228. The MoD 00-55 and MoD 00-56 will be excluded from consideration due to their focus on the United Kingdom's military requirements which, although similar to those of the United States, are not completely applicable. A specific study of IEC 61508-3 and NASA-STD-8719.13 are found later in this chapter, as both are highly applicable to the current research problem.

Software requirements frameworks focused on software safety tend to be highly customized to the environment, rather than being generic models; although attempts have been made to define a generic software safety requirements framework, these attempts have not been successful (Bhansali, 2005). General criteria for a software safety requirements framework have been identified by Bhansali (Bhansali, 2005). The general subset of requirements has been identified by examination of known software safety standards. Table 5 indicates the required elements identified in order to establish what Bhansali describes as the minimum subset of requirements needed to generate a one-size-fits-all software safety requirements framework. These requirements were identified by examination of standards from across all areas of industry, government, and safety-critical applications. Though Bhansali identified the specific required elements for such a generic framework, he did not make any determination of how these elements should be implemented. Bhansali's model of a generic requirements framework indicated five levels of security, with different levels required for each of these models; the assumption was that there would be different requirements per level of safety, indicating different

requirements for safety standards and specifications. The application domain would determine in most cases which of these requirements was needed at which level (Bhansali, 2005). The requirements for each level are identified within this research; however, the requirements at each level would need to be determined by the overall requirements of the system in question, rather than through a generic modeling process. (Bhansali, 2005).

Functional or preliminary hazard assessment	System safety assessment
Software requirements Validation	Special software Architecture
Safe design subset	Safe code subset
Traceability analysis	Independent code Analysis
Derived requirements Validation	Equivalent class testing
Boundary value testing	Machine instruction Coverage
Machine branch Coverage	Data set/use analysis
Control flow analysis	Stack analysis
Timing analysis	Numeric analysis
Complexity Measurement	Software quality Assurance
Software configuration Management	Software data load Management
System safety Verification	

Table 5: Required Elements for a Generic Software Safety Requirements Framework (From Bhansali, 2005)

A truly generic model has not yet been established to drive the construction of software safety in any application domain. A number of models that increase the generality of existing models or provide a general model that can be used to identify the safety requirements of a given system have been constructed. One recent model which integrates the factors, criteria, and models (FCM) approach of McCall and Boehm (more commonly used in quality analysis of software that is not highly safety-aware) was constructed by Medikonda and

Panchumarthy (Medikonda & Panchumarthy, 2009), and is demonstrated in the figure below. As can be seen in the system, the primary interaction with the requirements process within the framework is the completeness of requirements (based on system hazard analysis), and the identification of safety critical requirements is the main area in which criteria regarding software requirements interact. Many distinctions between levels of safety requirements are used in the model. These levels include safety requirements, which specify how safe the system should be (identified in many models by safety levels, as noted by Bhansali); safety-significant requirements, or functional and other quality requirements for safety requirement achievement; safety system requirements, which are requirements for internal safety systems such as automated shutoff switches, fire protection systems, etc; and safety constraints, or requirements for use of specific safety systems (Medikonda & Panchumarthy, 2009). Appropriate identification within the requirements-setting stage is key in Medikonda and Panchumarthy's model for identifying the requirements for software safety quality. Medikonda and Panchumarthy's model has not yet been placed into wide use, and stands as a potential generic model rather than a tried and tested one.

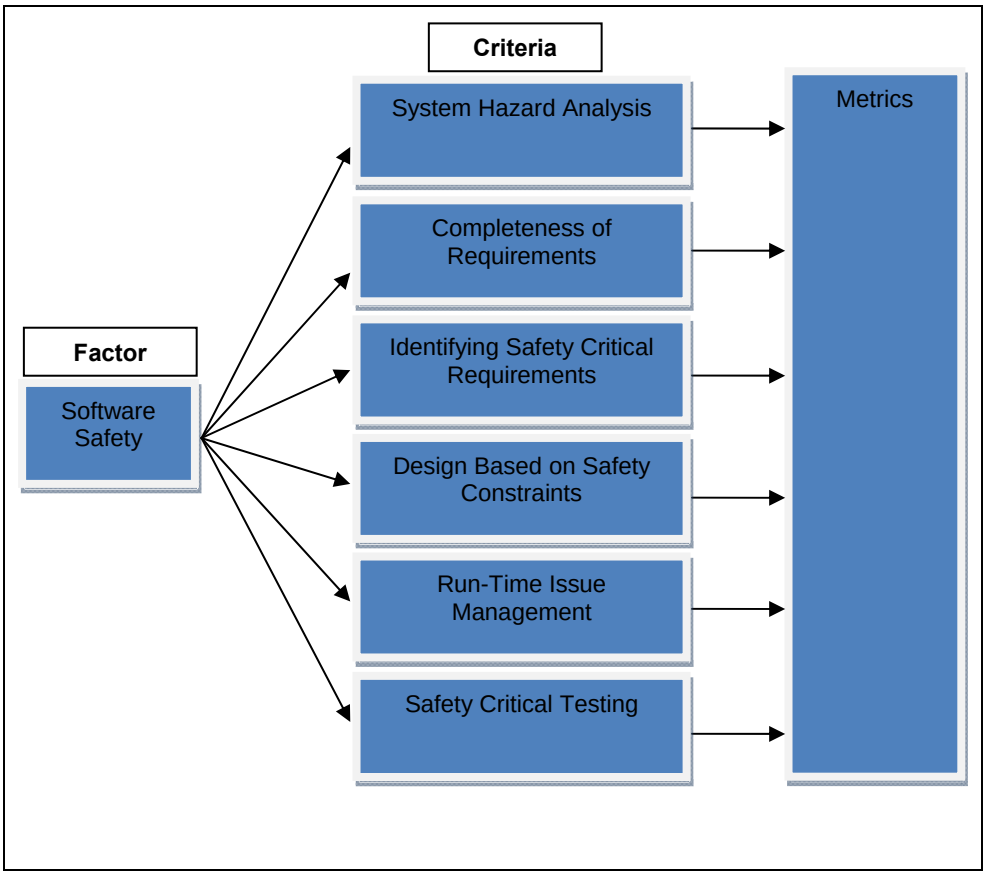


Figure 5: Software Safety Framework (After Medikonda & Panchumarthy, 2009)

L. NASA SOFTWARE SAFETY STANDARD (NASA-STD-8719.13)

One of the most comprehensive software safety requirements frameworks available is the NASA standard NASA-STD-8719.13 and its accompanying support materials and frameworks. The standard is applied to all software used in NASA (NASA, 2009), which makes the comprehensive software safety standard particularly applicable to the SSSTRP domain. The NASA standards for software safety have emerged from examination of the causes and effects of aerospace accidents, and determination of requirements for software safety that have emerged from the area (NASA, 2009). NASA-STD-8719.13 is based on the NASA Safety Manual (NPR 8753.3), which identifies the characteristics of safe systems and describes how these systems can be appropriately identified (NASA, 2009). The standard is accompanied by a guidebook, NASA-GB-

8719.13, which offers information on how the standard should be applied within the process of software engineering and requirements determination. The NASA standard is intended to apply to custom-engineered software, commercial off-the-shelf (COTS), modified off-the-shelf (MOTS), and government off-the-shelf (GOTS) software (National Aeronautics and Space Administration, 2004). The NASA standard is one of the most fully-featured software safety requirements available.

NASA 8719.13 identifies software safety requirements starting in the conceptual phase of the software design or acquisition process (National Aeronautics and Space Administration, 2004). The 8719.13 document purpose is described as being "to provide requirements to implement a systematic approach to software safety as an integral part of the project's overall system safety program, software development and software assurance processes" (National Aeronautics and Space Administration, 2004). Process and technical requirements for system safety are included in the description. Requirements for identifying safety-critical applications and systems that will impact these safety-critical applications, project management, planning and control activities, life cycle analysis, and software safety throughout the software life cycle are addressed; also identified are areas that would require modified approaches to software safety, such as COTS, MOTS, or GOTS systems (National Aeronautics and Space Administration, 2004). Legacy systems and the regulations for ensuring that these systems adhere to current safety standards and requirements are addressed (National Aeronautics and Space Administration, 2004).

The NASA standard contains a comprehensive discussion of how to determine whether or not a given system is safety-critical. For the evaluation, it uses guidelines including factors such as the cause or contribution of a hazard, hazard control or mitigation, and processing safety-critical commands or data (National Aeronautics and Space Administration, 2004). The detailed application behavior identification approach is intended to provide a complete risk assessment of how the software will be used, as well as what other requirements

exist for its determination. The process of identifying software safety requirements is performed through a preliminary hazard analysis (PHA), or risk assessment process, which examines the role of the software within the overall system. Software evaluation occurs during the conceptualization phase, before the planning for custom software or acquisition of non-custom software begins (National Aeronautics and Space Administration, 2004). The process of the PHA involves identifying hazards for specific requirements or system design choices for the software, and an overall system safety analysis (National Aeronautics and Space Administration, 2004). These analyses are then used to construct specific safety requirements for the software in terms of functionality and contextual placement within the system as a whole (National Aeronautics and Space Administration, 2004). These requirements are designated as software safety requirements, which are then integrated into the design or acquisition process alongside other functional and nonfunctional requirements for the software (National Aeronautics and Space Administration, 2004). A software safety plan is established and is maintained alongside the software as a record of the safety choices that were made during the conceptual stage of the design process. The model identifies archival processes that should be undertaken. The accompanying Guidebook can be used to operationalize the standard within the organizational environment; although the Guidebook is specific to NASA's organizational and development structure, much of the information within it is applicable to the naval gun weapon system domain.

M. IEC 61508-3

IEC 61508-3 is the IEC standard subsection that identifies the process of requirements determination for safety-critical applications (Medikonda & Panchumarthy, 2009). Although IEC 61508 was only published between 1998 and 2000, it had been in development since the mid-1980s through a Task Group designed to assess the challenges involved in ensuring software safety in programmable electronic systems (PES); these systems include computers and real-time embedded systems (Bell, 2006). There are currently eight identified

parts of IEC 61508, including Functional Safety and IEC 61508; General Requirements; Requirements for Electrical, Electronic and Programmable Electronic Systems; Software Requirements; Definitions and Abbreviations; Examples of Methods for the Determination of Safety-integrity Levels; Guidelines on the Application of parts two and 6; and Overview of Techniques and Measures (Bell, 2006). Part 3 (Software Requirements) holds the normative requirements (indicated by “shall”) that are applicable (Bell, 2006).

As in NASA-8719.13, IEC 61508’s safety requirements determination process contains a preliminary evaluation of the requirements for the system design (Bell, 2006). The focus is safety, as determined at the functional specification level, since research has indicated that the functional specification process is where the majority of safety-related failures in software occur (Bell, 2006). IEC 61508 is built on four safety integrity levels, which identify potential failure points and identify measures for overcoming the potential for failure within these systems (Bell, 2006). These safety integrity levels are identified through the probability of failure, although these identifications are different depending on the level of the function’s demand and/or continuous operational mode (Bell, 2006). In the case of a low-demand software system or component, the probability is defined as the probability that the component will fail to perform when demanded, while for high-demand and continuously operating systems, the definition is the probability of a dangerous failure per hour (Bell, 2006). IEC 61508 takes a risk-based approach to determining software safety, identifying the potential outcomes of a failure as well as its probability in order to determine whether a design is acceptable or unacceptable in terms of safety (Bell, 2006).

IEC 61508 identifies requirements determination for software safety requirements and includes a complete software life cycle approach to determining software safety in the overall case (Bell, 2006). Figure 5 demonstrates the life cycle approach in detail.

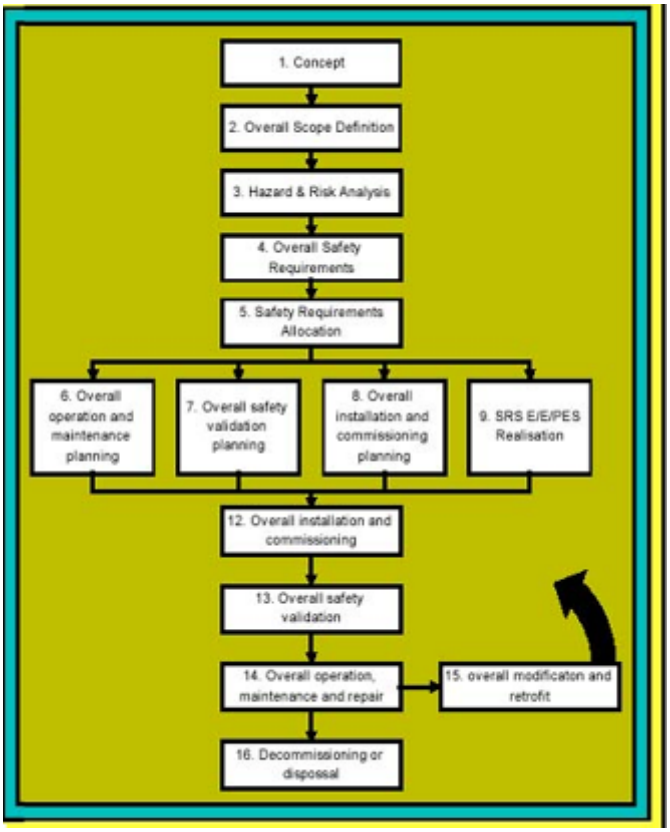


Figure 6: IEC 61508 Life Cycle Framework (From Bell, 2006)

As demonstrated by Bell's IEC 61508 Life cycle Framework, the focus on safety requirements is during the conceptual process and before integration. Unlike the NASA standard, little attention is paid to off-the-shelf software or the modification of legacy software, which could be rectified by modification of the framework structure in order to meet the needs of the current research.

Although IEC 61508 is presented as a universal standard for software safety requirements, the framework lacks focus and features for other areas of software design (vitally, in this case, excluding military applications), and "the approach taken is 'do it all' or to justify not doing it at all" (Bhansali, 2005). Thus, IEC 61508 does not meet the requirements for naval gun weapon system software safety evaluation. The IEC 61508 standard does not directly apply to the naval gun weapon system SSSTRP environment; however, it has been used successfully within the military system environment in the past. Although IEC

61508 is a paid standard, it has a record of positive application, and it is a carefully designed standard that can be modified to meet many of the needs of the current project.

In summary, the IEC 61508 standard is less complete than NASA 8719.13 (National Aeronautics and Space Administration, 2004), as it does not contain requirements or specifications for functional safety or safety verification requirements, which decreases the scope of safety requirements determination it offers (Bell, 2006). These potential disadvantages do not remove the IEC 61508 standard from consideration for use in the naval gun weapon system domain, but do reduce its utility and increase the amount of difficulty involved in the system's use.

N. SUMMARY

The literature review discusses the current state of methodology and tools within the software safety domain, with specific focus on enterprise systems. This review contained information about COTS integration risks, vendor selection, software acquisition, software architecture, the Navy's Open Architecture Enterprise Program, abstract modeling methodologies, and software safety standards and frameworks.

The literature review has demonstrated the need for a modeling methodology that can model the system's interaction with the environment. Additionally, a capability gap exists that enables the SSSTRP evaluation team to accomplish an evaluation of both functional and nonfunctional requirements, such as performance aspects, speed of execution, and other software safety quality of service indicators. The next chapter addresses the details of a solution to the problem of pre-acquisition software safety analysis using the Monterey Phoenix (MP) modeling methodology.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SYSTEM ARCHITECTURE MODELING METHODOLOGY FOR NAVAL GUN WEAPON SYSTEM SOFTWARE

A. INTRODUCTION

The Introduction chapter explains the specific problems associated with the SSSTRP naval acquisition process, and the concept of the system architecture modeling methodology that was developed to address these problems. This chapter also contains demonstrations of prototype software that implements the modeling methodology, as well as test cases using a naval gun weapon system. Finally, this chapter contains a suggested prototype SSSTRP evaluation methodology that describes how the tools may be implemented within the current SSSTRP process.

Providing the SSSTRP community with high-level models that may satisfy a portion of the software safety assessment process improves the current inspection-based evaluation methodology. Without a high-level modeling process, the alternative is to implement the system and to perform testing. Manual testing is a very expensive and timely alternative, which may be partially satisfied using the prototype methodology and tools that are covered in this chapter.

The proposed SSSTRP evaluation methodology and tools that are demonstrated in this chapter improve the SSSTRP evaluation process in the following ways:

- Identify unintended system behaviors
- Provide a high-fidelity system safety assessment
- Tools for evaluating nonfunctional requirements
- Perform assessments at appropriate levels of abstraction

The goal for the gun weapon system case study is to test a proposed modeling tool in order to improve the current state of the SSSTRP evaluation process.

B. DESCRIPTION OF A NAVAL GUN WEAPON SYSTEM

The U.S. Navy gun system diagram used for this research was provided by the U.S. Navy's Weapons Explosive Review Board (WESERB) as part of the documentation that accompanies the research in Chapter I. The gun weapon system contains 17 separate systems, all connected through a single network. The gun weapon system was modeled using MP event grammar. The modeling application, herein referred to as "Eagle6," is the product of this research. The Eagle6 application accepts MP modeling language and gives the user the ability to write formal queries that return specific sets of scenarios. For the purposes of defining limitations and definitions of scope, we have defined Scope as the number of model iterations.

Eagle6 (explained in detail later in this dissertation) uses an exhaustive and probabilistic approach to generating scenarios, and has the following capability:

- Eagle6 is based on executable models and is able to generate all possible scenarios within a given scope.
- Eagle6 provides a high-level abstraction of the interaction between a software system and its environment.
- Eagle6 supports multiple views of system architecture that are generated from the same MP model.
- Eagle6 supports random scenario generation for statistical evaluation.

The following is a description of the systems in the Gun weapon system model, with the model abbreviation in brackets:

Systems Included in the Gun weapon system Model:

- C&D [CD]-Command and Decision. The software system that performs all functions within the Aegis combat system
- AN/SPS-67 [R2D]-2-D Surface Search Rotating Radar

- AN/SPY-1D [R3D]-3-D Air Defense and Surface Search Phased Array Radar
- Gun Mount Processor AN/UYK-44 EP/OSM [GMP]-One sub-element of the GCS, which takes information from the GCC and provides services to the gun mount
- Gun Console Computer [GCC]-Sub-element of the GCS. It interfaces with Aegis and other ship sensors and performs fire control calculations and provides data to the GMP.
- Optical Sight System MK 46 Mod 1-Control Display Console MK 132 Mod 0 [CDC]-The operator console used to control the MK46 Optical Sight
- Optical Sight System MK 46 Mod 1-Electro-Optic Director MK 85 Mod 1 [EOD]-The Optical Sight director system (installed above the bridge) that rotates and elevates per operator's commands. The TV, IR, and laser range finder sensors are installed on the director, which points them in the right position
- Gun Mount Control Panel MK 437 Mod 1 [GMCP]-Backup Operator's console installed below the gun mount. It is used in case the main ADS console in the combat information center goes down.
- Gun Mount EX 45 Mod 4 [GM]-The 5" gun mount. Holds 20 rounds in the drum and fires 18-20 rounds per minute.
- AEGIS Display System [System_ADS]-The software that drives all displays and console operator actions within the ship's Aegis combat system (The operator interface software to C&D).
- ACTS [System_ACTS]-Command and Control Backup Module

- AEGIS Clock/Gyro Data Converter Cabinets (2) [System_ACGDCC]-System that provides time and ship's attitude information to C&D.
- FODMS [System_FODMS]-Data Multiplexing System.
- Gun Computer System [System_GCS]-System used to perform all core gun fire control functionality.
- Recorder/Reproducer MK27 Mod 1 [System_Recorder]-Part of the GCS that is responsible for loading operational program data and recording GCS data.
- Velocimeter MK 5 Mod 0 [System_Velocimeter]-The sensor (radar) on the MK 45 Gun Mount that monitors outgoing projectiles after firing and calculates projectile velocity, used to improve fire control accuracy.
- Control Panel EP2 MK 281 Mod 9 [System_CP]-The electronic panel that is used to turn on, set up, load, and locally control the gun mount. It is separate from the GCS.

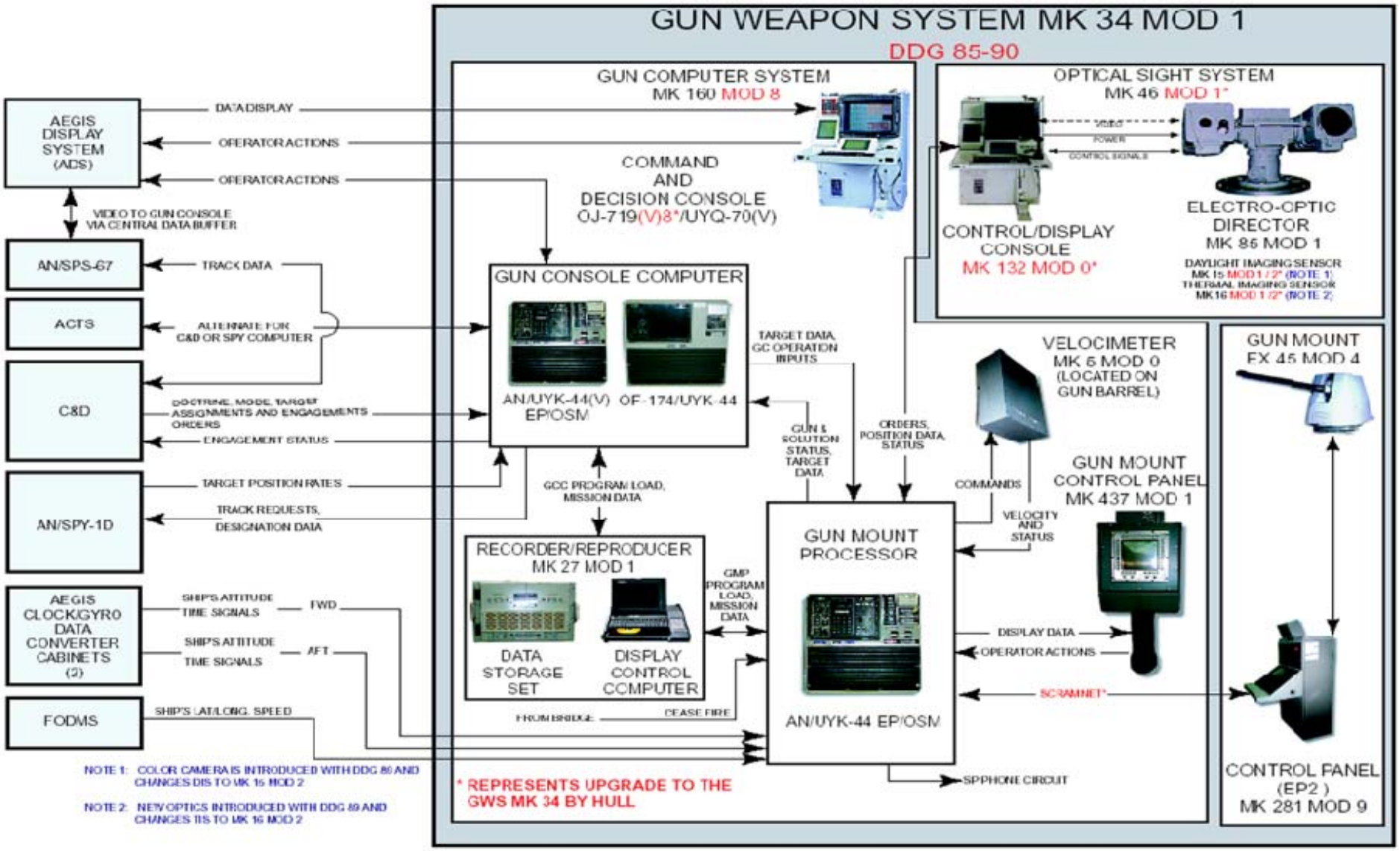


Figure 7: From Gun Weapon System MK 34 Mod 1 (From Naval Gunnery Project Office PEO IWS3C)

C. IDENTIFICATION OF PROBLEMS FOUND IN THE PRE-ACQUISITION SOFTWARE SAFETY EVALUATION PROCESS

1. Domain-Specific Issues Covered in This Research

The primary responsibility of the SSSTRP is to identify possible hazard states when evaluating a proposed gun weapon system change. Our research showed many areas of the SSSTRP process that warrant attention, but the focus of our research was narrowed in order to enable us to focus on the following critical issues:

- Testing domain architecture models for software safety violations–The goal of this research is to provide a solution that enables the SSSTRP to automatically generate a number of scenarios that test for software safety violations.
- Estimation of software performance based on architecture models–The goal of this research is to create tools that enable the SSSTRP to test nonfunctional requirements using Formal Methods. The tool answers the question, “How will the software behave once it is a part of our system?”

2. Domain-specific Issues Not Covered in This Research

- Software inspection techniques
- SSSTRP structure and evaluation methodology
- Vendor Technical Data Package design/structure
- System functionality overlap in Open Architecture (OA) environments
- Development of more specific and effective guidelines for how to test safety aspects of COTS software
-

D. OVERVIEW OF THE MONTEREY PHOENIX METHODOLOGY

MP was chosen for this research because it satisfies three primary needs for this domain:

- MP has the ability to model nonfunctional requirements. Testing how a system interacts with the environment is a critical need that has not been available to the SSSTRP.
- MP has the ability to evaluate formal Assertions. Because MP results are obtained from an exhaustive generation of all scenarios within scope, determining Hazard States enables the SSSTRP to evaluate potential system changes with greater effectiveness.
- MP has the ability to extract visual representations of scenarios, thereby yielding a result that is usable and readable by the layman. MP Modeling Definitions

MP is used in this domain to create a model with a set of architectural properties. Attributes are properties of an event that may be used to define domain model representations. Attributes are valuable as they represent a more detailed (and measurable) application state. The intention is to model the concept of an event state associated with the event, thereby enabling the ability to evaluate the model for predefined unsafe states.

1. MP Scenario (Event Trace)

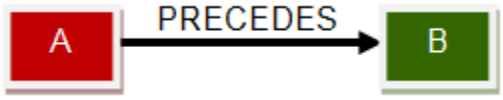
An Event is defined as any detectible action. A scenario is a set of events of different types and two sets of relationships between them (IN and PRECEDES).

A grammar rule has form:

A: right-hand-part, where A is an event type name. Event types that do not appear in the left hand part of rules are considered atomic.

Events are visualized by small circles, and basic relations by arrows:

Event Relationships IN and PRECEDES



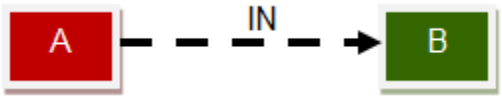


Figure 8: MP Event Trace

MP modeling requires a ROOT event that represents the starting point for a series of following relational events. In the following examples, R, A, B, C are events, and the event R is the ROOT event:

- R: {A B C} – Root event R contains UNORDERED events A, B, and C
- R: (A B C) – Root event R contains ORDERED events A, B, and C
- R: {* A *} – ROOT event R may have zero or more UNORDERED events A
- R: (* A *) – ROOT event R contains zero or more ORDERED events A
- R: [A] – ROOT event R may contain optional event A
- R: (A | B | C) – ROOT event R contains either A or B or C

The following MP construct definitions explain the use of the MP language:

2. Unordered Events: R: {A B C}

Event R contains events A,B,C. Events A,B,C are not ordered (no precedes relationship between them).

Optional Event Traces for R: {A B C}

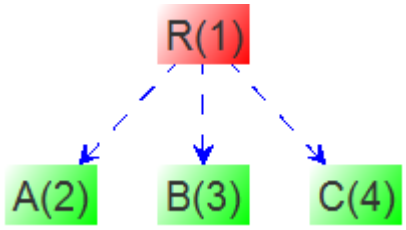


Figure 9: MP Unordered Events: R: {A B C}

3. Ordered Events: R: (A B C)

Event R contains events A, B, C. Events A, B, C are ordered: A precedes B, B precedes C;

Event Traces for R: (A B C)

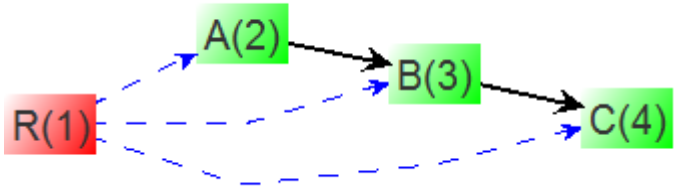


Figure 10: MP Ordered Events: R: (A B C)

4. Multiple Unordered Events: R: {* A *}

The * is used to allow the modeler to describe an event that happens zero or more times. Given an expansion scope of n, event R has (n+1) scenarios.

Event Traces for R: {* A *}

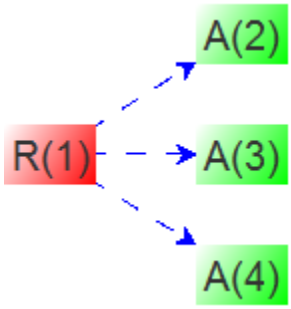


Figure 11: MP Multiple Unordered Events: R: {* A *}

5. Multiple Ordered Events: R: (* A *)

This sequence denotes a set of zero or more events of type A with an ordering relation between them. Given an expansion scope of n, event R has (n+1) scenarios.

Event Traces for R: (* A *)

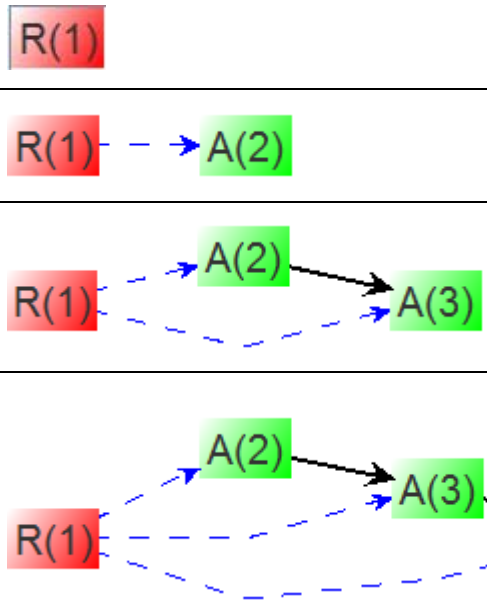


Figure 12: MP Multiple Ordered Events: R: (* A *)

6. Optional Events: R: [A]

This sequence denotes an optional event A. Event R has two scenarios: one scenario where R is empty, and one scenario where R contains A.

Optional Events for R: [A]

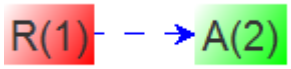


Figure 13: MP Optional Events: R:[A]

7. Alternative Events: R: (A | B | C)

Alternative events are denoted by separating events by using vertical bars. The following example contains three alternative events, event B, event C, or event D. Event R has three scenarios. One scenario where R contains A, one scenario where R contains B, and one scenario where R contains C:

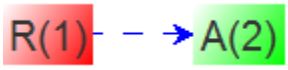
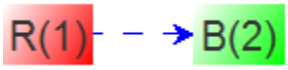
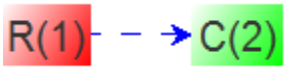
Optional Event Traces for R: (A | B | C)




Figure 14: MP Alternative Events: R: (A | B | C)

8. Introduction of SHARE ALL Construct and Constraints

The construct SHARE ALL is used to describe event coordination and system constraints. The SHARE ALL construct identifies events that can be shared by other events. The following MP model contains two components TaskA and TaskB with a connector between them. A Connector enables components to interact, for example send and receive a message, call each other and pass a parameter, or use a shared memory to deliver a data item. The schema Send_Receive_Activity specifies the behavior of components involved in a single transaction.

SCHEMA Send_Receive_Activity

ROOT TaskA: (Send);
ROOT TaskB: (Receive);
ROOT Connector: (Send Receive);

TaskA, Connector **SHARE ALL** Send;
TaskB, Connector **SHARE ALL** Receive;

The rule section introduces Root events TaskA, TaskB, and Connector, while Send and Receive events are needed to specify the root event's structure. The event type stands for a set of event traces satisfying the event structure defined for that type. The constraints section uses the predicate **share all**, which is defined as X, Y are root events, and Z is an event type:

X, Y **SHARE ALL** Z ≡ { v: Z | v IN X} = {w: Z | w IN Y}

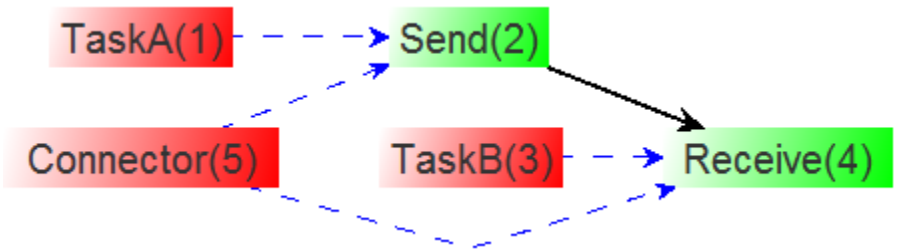


Figure 15: Scenario Generated from MP Schema_Send_Receive_Activity

The events are represented by rectangles (red rectangles are ROOT events, and green rectangles are non-ROOT events), and the relationships are represented by arrows (blue dashed arrows are IN relationships, and black solid line arrows are PRECEDES relationships).

The example of MP contains:

- TaskA(1), Connector(5), and TaskB(3) are ROOT events
- Send(2) and Receive(4) are non-ROOT events of type TaskA
- Receive(4) is a shared event of TaskB and Connector(5)
- Send(2) PRECEDES Receive(4)
- Connector(5), TaskB(3) are IN Receive(4)

9. MP Attributes

There are two types of attributes: static and dynamic. Static attributes are values that are set at the beginning of a model and do not change. Dynamic attributes have a value that may change in different parts of the scenario. The Eagle6 prototype uses static attributes that enable query language. Dynamic attributes are reserved for future research (Auguston & Whitcomb, System architecture specification based on behavior models, 2010).

10. MP Expansion Scope Construct

Scope is defined as the number of model iterations. The purpose of the Expansion Scope is to limit the size of the "*" rule in order to better define the scenario's parameters. For example, if the test scenario requires the gun weapon system to fire three rounds, the scenario's scope is set to "3," thereby removing the infinite ("*") default parameter. In the absence of an Expansion Scope that is detailed in the model design, setting this value will result in a finite number of scenarios.

MP language by setting an expansion scope each time the “*” rule is defined as:

(* <m-n> *)

<n> is considered an abbreviation for <0-n>

(* <0-n> event *) and { * <0-n> event * }

Becomes

(* <n> event *) and { * <n> event * }

11. Example MP Model

The following example MP code contains a scenario that contains naval guns, with each gun firing at a target. The test scenario represented in MP is as follows:

- A minimum of 1 gun system, and a maximum of two gun systems
- Each weapon can fire zero, one, or two times, maximum.
- The result of the test can be Hit or Miss

SCHEMA: GWS_SSSTRP_Test

ROOT GWS_Cycle_Test: { * <1-2> Gun_System *};
Gun_System: (* <2> Shoot *);
Shoot: (Load Fire (Hit | Miss));

The MP code is described as follows:

MP Code: "ROOT GWS_Cycle_Test: { * <1-2> Gun_System *};

Description: The initiating event (ROOT) is called the GWS_Cycle_Test. The GWS_Cycle_Test event has 1-2 Gun_System events.

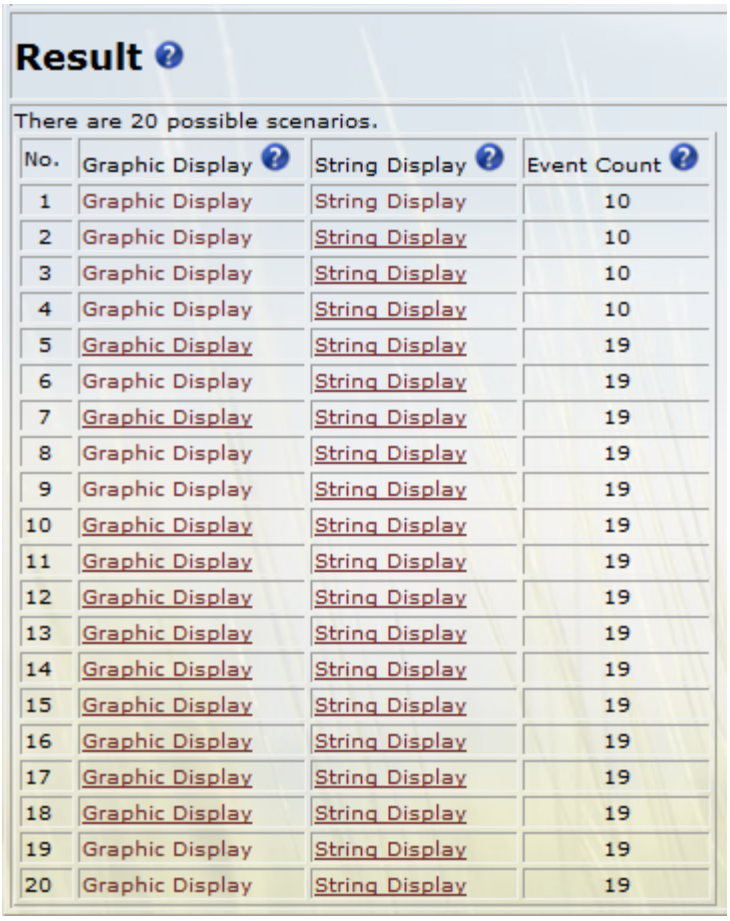
MP Code: "Gun_System: (* <0-2> Shoot *);

Description: The Gun_System has zero, one, or two Shoot events.

MP Code: "Shoot: (Load Fire (Hit | Miss));"

Description: The Shoot event has one event that ends in a Hit or Miss event.

The MP code resulted in a total of 20 possible scenarios, with the scenario with the least amount of events being 10, and the largest being 19:



Result

There are 20 possible scenarios.

No.	Graphic Display	String Display	Event Count
1	Graphic Display	String Display	10
2	Graphic Display	String Display	10
3	Graphic Display	String Display	10
4	Graphic Display	String Display	10
5	Graphic Display	String Display	19
6	Graphic Display	String Display	19
7	Graphic Display	String Display	19
8	Graphic Display	String Display	19
9	Graphic Display	String Display	19
10	Graphic Display	String Display	19
11	Graphic Display	String Display	19
12	Graphic Display	String Display	19
13	Graphic Display	String Display	19
14	Graphic Display	String Display	19
15	Graphic Display	String Display	19
16	Graphic Display	String Display	19
17	Graphic Display	String Display	19
18	Graphic Display	String Display	19
19	Graphic Display	String Display	19
20	Graphic Display	String Display	19

Figure 16: MP Example: GWS_Cycle_Test Results

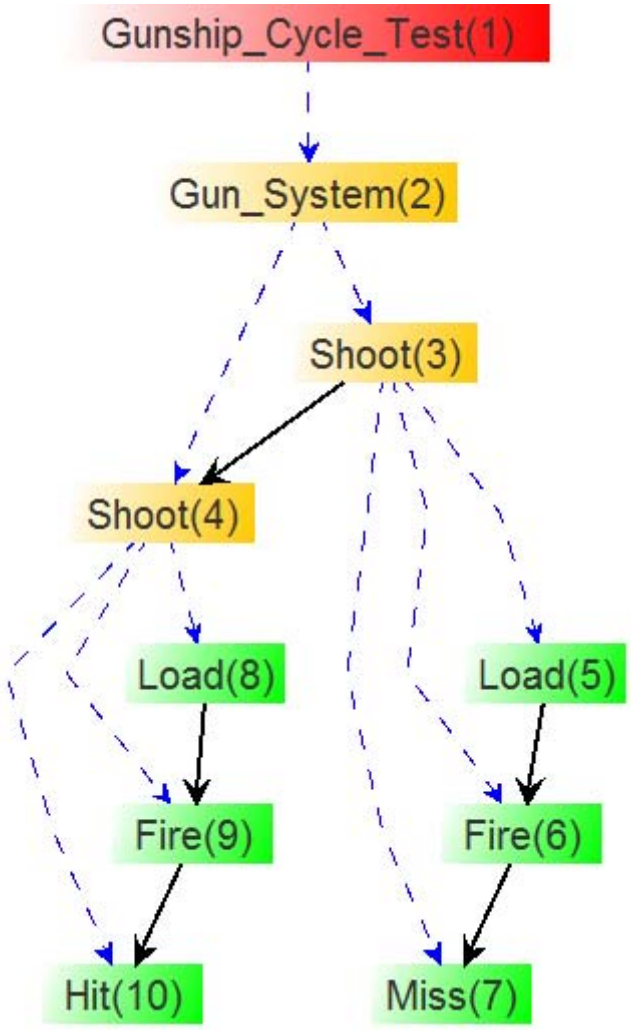


Figure 17: Scenario Generated from MP Schema: GWS_Cycle_Test #3

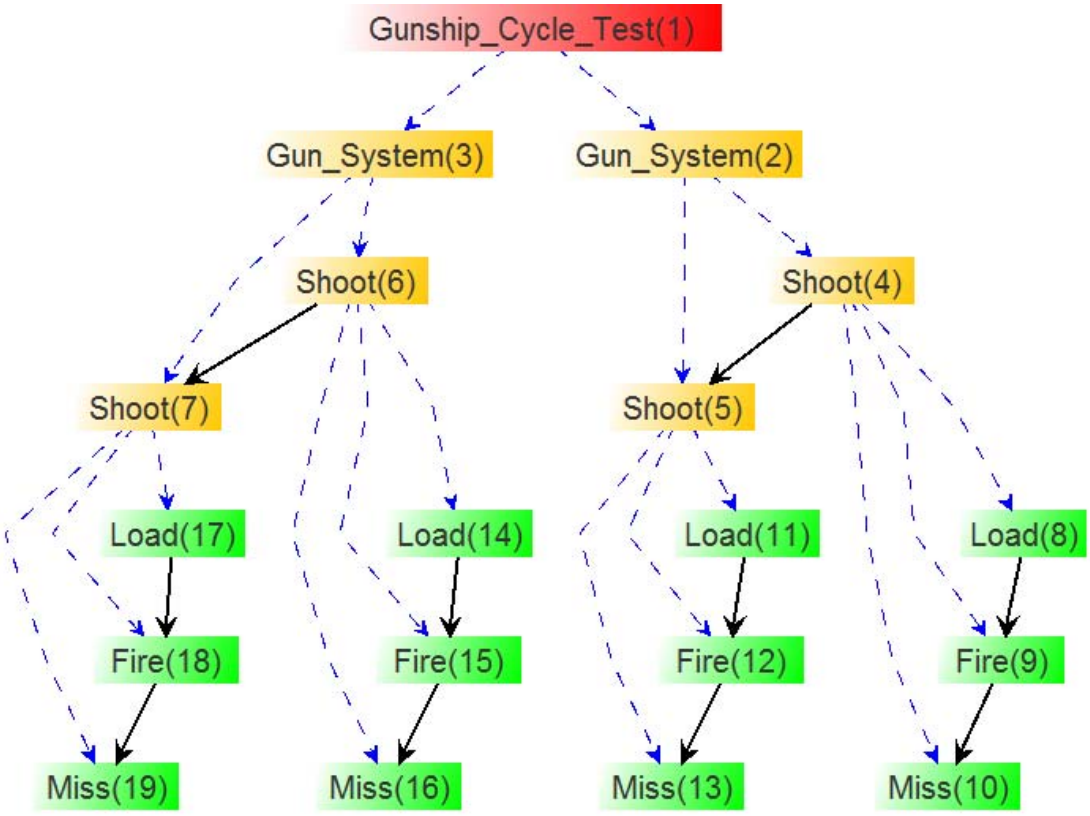


Figure 18: Scenario Generated from MP Schema: GWS_Cycle_Test Scenario #20

12. Small Scope Hypothesis

The gun weapon system model uses event grammar and includes the ability to execute exhaustive testing for scenario generation within scope (Auguston, Monterey Phoenix, or How to Make Software Executable, 2009), (Andoni, Daniliuc, Sarfraz, & Marinov, 2002). Our hypothesis of finding unsafe system states using a small scope size is based on Jackson's Small Scope Hypothesis. "Small Scope Hypothesis" (Jackson, Software abstractions: logic, language, and analysis, 2006) (Jackson & Damon, Elements of style: Analyzing a software design feature with a counter example detector, 1996) argues that a high proportion of bugs can be found by testing the system within a small scope of test cycles. The ability to introduce environmental events such as missiles, power outages, and system failure in small scope testing

(Auguston, Software architecture built from behavior models, 2009) showed that MP is able to introduce critical environmental events that have a high probability of rendering the gun system unsafe.

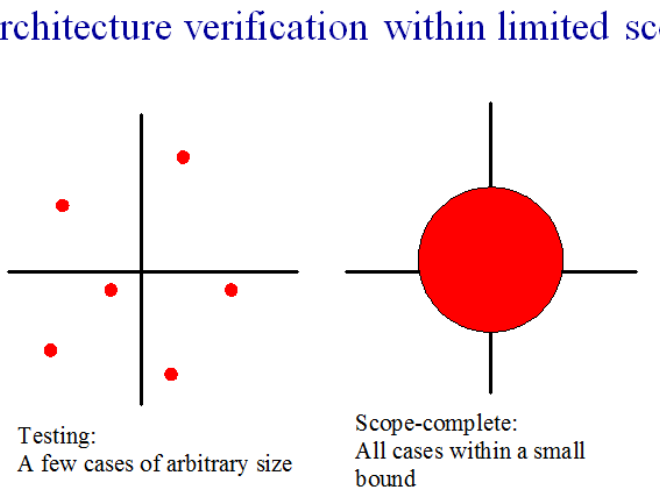


Figure 19: Jackson's Small Scope Hypothesis (After Jackson, Software abstractions: logic, language, and analysis, 2006)

Jackson's Small Scope Hypothesis that most errors can be demonstrated on small counter examples is demonstrated in Eagle6. Eagle6 has two primary means for evaluating software safety using relatively small scope sizes:

Exhaustive Search – Exhaustive search is the process of generating all possible scenarios from the MP model up to a given scope, and querying the result set. The Exhaustive Search enables the user to find scenarios that produce counter-examples of assertions.

Random Approach – Random approach is designed to generate random scenarios within scope to calculate statistical estimates. The purpose of this functionality is to create estimates that are used for software safety assessments.

Summary: Jackson's Small Scope Hypothesis graph represents the idea that an exhaustive test within a small scope is much better than an unstructured test with arbitrary test parameters.

13. Use Case Representation in MP

The following demonstration includes a simple gun weapon system use case and the corresponding MP model. The purpose of the demonstration is to show that MP has the capability to extract use cases from an MP model, thereby creating the capacity for formal testing.

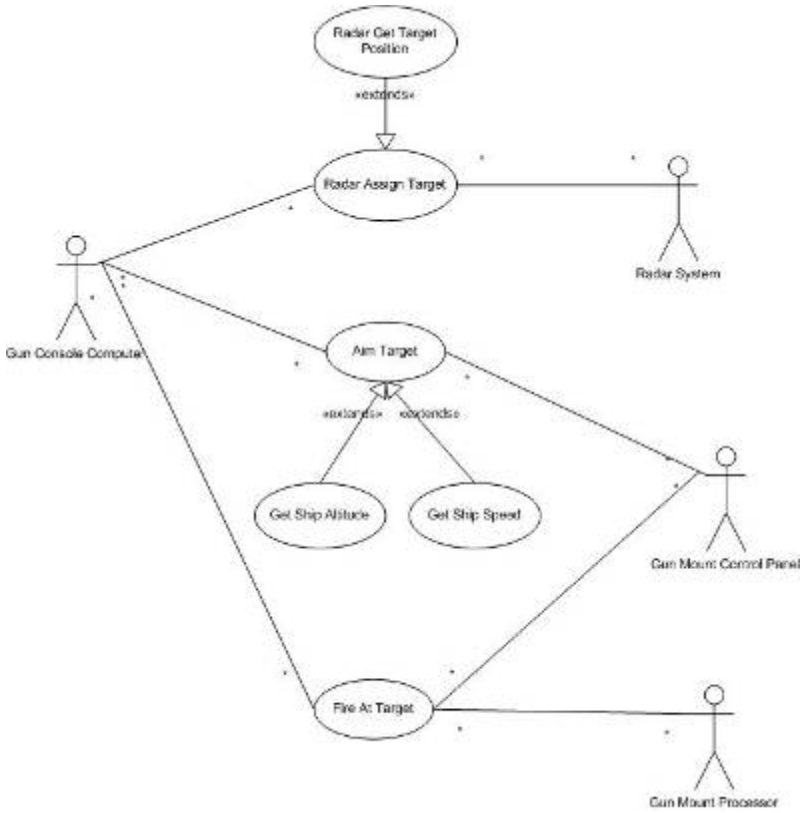


Figure 20: Gun weapon system Fire Use Case Diagram in UML Notation

UML Actors:

- Gun Console Computer [GCC] - Sub-element of the GCS. It interfaces with Aegis and other ship sensors and performs fire control calculations and provides data to the GMP.
- Radar System [R3D] - 3D Air Defense and Surface Search Phased Array Radar

- Gun Mount Control Panel [GMCP] - Backup Operator's console installed below the gun mount. It is used in case the main ADS console in the combat information center goes down.
- Gun Mount Processor [GMP] - One sub-element of the GCS, which takes information from the GCC and provides services to the gun mount

Use Cases:

- Radar Get Target Position - Uses the Radar information to get the target position.
- Radar Assign Target - Uses information from the Radar and Gun Console Computer to assign the target.
- Get Ship Altitude - Calculates the ship's current altitude.
- Get Ship Speed - Calculates the ship's current speed.
- Aim Target - Uses the information from the Gun Mount Control Panel Actor and Gun Console Computer Actor, as well as the Get Ship Altitude and Get Ship Speed Use Cases to set the gun aiming function.
- Fire At Target - Uses information from Gun Mount Processor and Gun Console Computer to execute a fire command.

14. Use Case MP Model

```
ROOT GunConsoleComputer_activity: {
        RadarAssignTarget
        AimTarget
        FireAtTarget
};
```

```
ROOT RadarSystem_activity: {
        RadarAssignTarget
};
RadarAssignTarget: {
        RadarGetTargetPosition
};
ROOT GunMountControlPanel_activity: {
        AimTarget
        FireAtTarget
};

AimTarget: {
        GetShipAltitude
        GetShipSpeed
};
ROOT GunMount_activity: {
        FireAtTarget
};
```

GunConsoleComputer_activity, GunMount_activity SHARE ALL FireAtTarget;

GunConsoleComputer_activity, RadarSystem_activity SHARE ALL RadarAssignTarget;

GunConsoleComputer_activity, GunMountControlPanel_activity SHARE ALL AimTarget, FireAtTarget;

The following figure represents a scenario generated from the MP model:

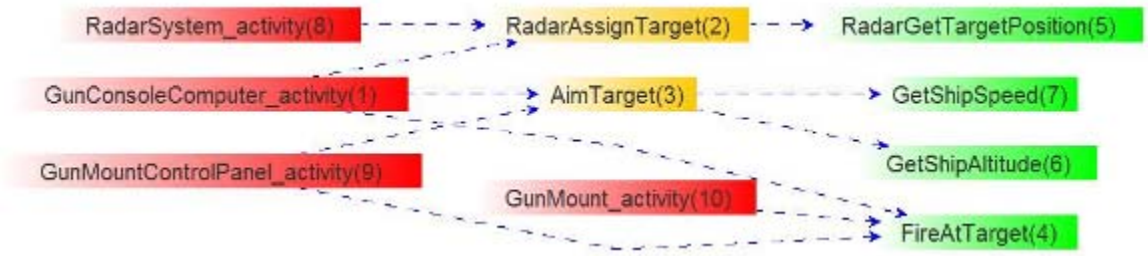


Figure 21: Example of Use Case Modeling via MP

In UML, Uses Case designs may contain conditional nodes. Use Case views generated by MP are single views of Use Case scenarios which clarify potential system behavior.

15. Evaluation of MP

MP has several features that apply to the gun weapon system software safety domain:

- MP provides a high level of abstraction–The MP modeling methodology has the capability to model system behavior at the abstract level without any detailed information about the specific system (Rivera, 2010). This attribute allows for testing and debugging earlier in the acquisition life cycle, as there is no need to continue the acquisition process if safety-related issues are found during the initial stages of evaluation.
- MP supports continuous refinement–The ability to insert an event such as a missile strike, power outage, or any other environmental event is critical for testing a potential system change. Systems work well in the lab. MP allows for the ability to test using an environment model. The ability to bring together the environment and the system in the same model is a new development. MP

allows for this new capability (Auguston & Whitcomb, System architecture specification based on behavior models, 2010).

- The MP framework provides high-level abstractions that may be used to analyze system behavior by checking assertions (Rivera, 2010). Having the ability to quickly test a potential system change without needing specific system details streamlines the acquisition process while increasing the fidelity of the evaluation process.
- The Use Case example demonstrates that MP supports the ability to generate and extract different views from an MP model. The ability to provide stakeholders graphical representations of potential scenarios that may end in a hazard state is necessary. Additionally, because MP supports formal methods, testing using assertions has a high level of fidelity, given that the model does an exhaustive search for all counter-assertions within scope.

MP provides the means to describe environmental behavior, which is why it was chosen as the modeling tool of choice for the "Eagle6 Prototype Software Architecture Modeling Software," which is described later in this dissertation.

E. PROTOTYPE NAVAL GUN WEAPON SYSTEM MODEL

The gun weapon system model found in Appendix A is a model written entirely in MP. It utilizes attributes in order to enable the evaluation of system properties.

1. The Purpose of the Naval Gun Weapon System Model

The design of the gun weapon system model is meant to satisfy the following two requirements:

- Assertion-checking via an exhaustive generation of scenarios within scope.
- Defining model properties in order to determine the probability of a particular scenario.

2. Introduction to the Model

The gun weapon system model found in Appendix A is comprised of all system components identified in Chapter III.A, "Description of Naval Gun weapon system." The model event begins with the R2D Radar identifying a target, and ends with a Gun Console Computer Open Fire command.

3. Gun Weapon System Model Properties

Each system in the model has a root event that describes the system activity. The following is a list of systems used in the gun weapon system model, with a list of included events that make up the ROOT activity. Also included in each section are the test results for Scope, Total Scenarios, and the Total Processing Time.

Scenario Generation Result Definitions:

- Scope – Total scenarios generated from Eagle6
- Total Scenarios – The total number of possible scenarios within the model scope.

Processing Time – The amount of time it took for Eagle6 to generate an exhaustive search for all possible scenarios within scope.

ROOT R2D_activity - the activity of AN/SPS-67 [R2D] - 2-D Surface Search Rotating Radar

- R2D_displayNewTarget - R2D displays a new target on the screen

MP Model:

ROOT R2D_activity: {*
R2D_displayNewTarget // *R2D displays a new target on the screen*
*};

Note: The notation represents unordered events that may happen simultaneously.

Model Results:

Scope: 1
Total Scenarios: 2
Processing Time: 0.01 Seconds

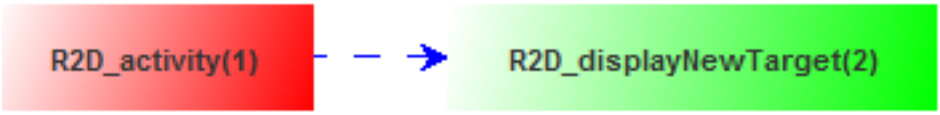


Figure 22: Scenario Generated from MP Schema: Gun weapon system Model R2D_activity

---

ROOT CD_activity - the activity of C&D [CD] - Command and Decision. The software system that performs all functions within the Aegis combat system

- CD_spotNewTarget - CD spots a new target on R2D screen
- CD_ignoreTarget - CD ignores target
- CD_request_GCC_setTarget - CD requests GCC to set target (requires more information about the target)
- CD_wait_GCC_setTarget - CD waits for GCC to set target
- CD_targetLost - CD loses target
- CD_followTarget - CD follows target movements and waits to see what happens
- CD_request_GCC_openFire - CD requests GCC to open fire at target
- CD_wait_GCC_openFire - CD waits for GCC to open fire

MP Model

ROOT CD_activity: {* CD_spotNewTarget *};

CD_spotNewTarget: (*// CD spots a new target and waits for R2D*

R2D_displayNewTarget *// R2D displays a new target*

(CD_ignoreTarget *// CD ignores target*

| (CD_request_GCC_setTarget *//Requests GCC to set target*

CD_wait_GCC_setTarget *// CD waits for GCC to set target*

((GCC_targetNotSet *// GCC fails to set target*

CD_targetLost) |

(GCC_targetSet *// GCC sets the target and returns target info*

CD_followTarget *// CD follows target and waits*

(CD_ignoreTarget *// CD ignores target*

| (CD_request_GCC_openFire *// CD requests GCC to open fire*

CD_wait_GCC_openFire *// CD waits for GCC to open fire*

(GCC_openFireFailed *// GCC failed to open fire*

| targetMissed *// target is missed*

| targetHit)))))))); *//target is hit*

//_____

R2D_activity, CD_activity SHARE ALL R2D_displayNewTarget;

//_____

Model Results:

Scope: 1

Total Scenarios: 7

Processing Time: 0.01 Seconds

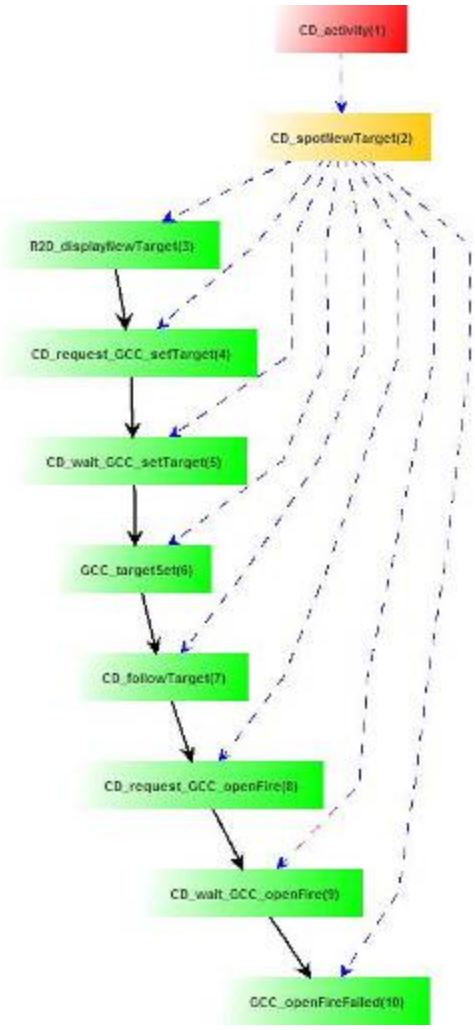


Figure 23: Scenario Generated from MP Schema: CD_activity Scenario #7

ROOT GCC_activity - The activity of Gun Console Computer [GCC]-Sub-element of the GCS. It interfaces with Aegis and other ship sensors and performs fire control calculations and provides data to the GMP

- GCC_setTarget - GCC sets a target (waits for CD to request setTarget and returns target information)
    - **GCC_targetNotSet** - GCC fails to work
    - **GCC_request_R3D_setTarget** - GCC requests R3D to set target (requires more information about the target)

    - **GCC_wait_R3D_setTarget** - GCC waits for R3D to set target
    - **GCC_targetSet** - GCC sets the target and returns target info to R3D
- GCC_openFire - GCC open fires on target (waits for CD to request openFire and opens fire)
    - **GCC_openFireFailed** - GCC is not working and it fails to open fire
    - **GCC_request_GMP_openFire** - GCC requests GMP to open fire
    - **GCC_wait_GMP_openFire** - GCC waits for GMP to open fire

Model Results:

Scope: 2

Total Scenarios: 157

Processing Time: 1.15 Seconds

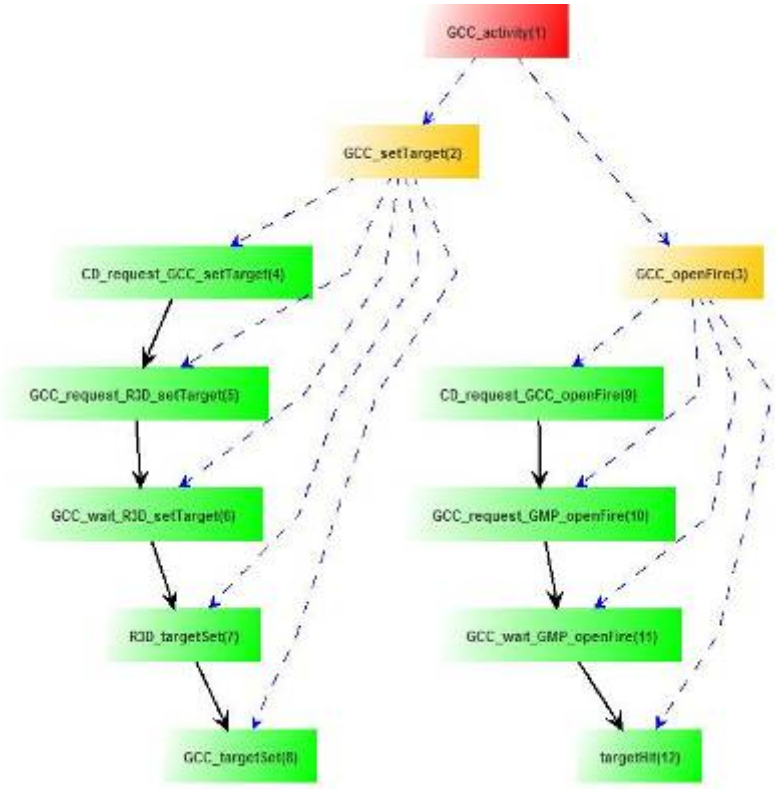


Figure 24: Scenario Generated from MP Schema: GCC_activity Scenario #13

ROOT GMP_activity - the activity of Gun Mount Processor AN/UYK-44 EP/OSM [GMP]-One sub-element of the GCS, which takes information from the GCC and provides services to the gun mount.

- GMP_answerRequest_GCC_openFire - GMP answers request from GCC to open fire (waits for GCC to request openFire and requests the same to GMCP)
  - **GMP_openFireFailed** - GMP is not working and it fails to open fire
  - GMP_answerRequest_GMCP_ossData - GMP answers a request from GMCP for optical sight target data (it waits for a request and it sends data)
    - **GMP_request_CDC_ossData** - GMP requests CDC oss data

- **GMP_wait_CDC_ossData** - GMP waits for CDC oss data
- **GMP_failReceiving_CDC_ossData** - GMP does not receive oss data from CDC
- **GMP_receive_CDC_ossData** - GMP receives oss data from CDC

Model Results:

Scope: 3

Total Scenarios: 1885

Processing Time: 2.14 Seconds

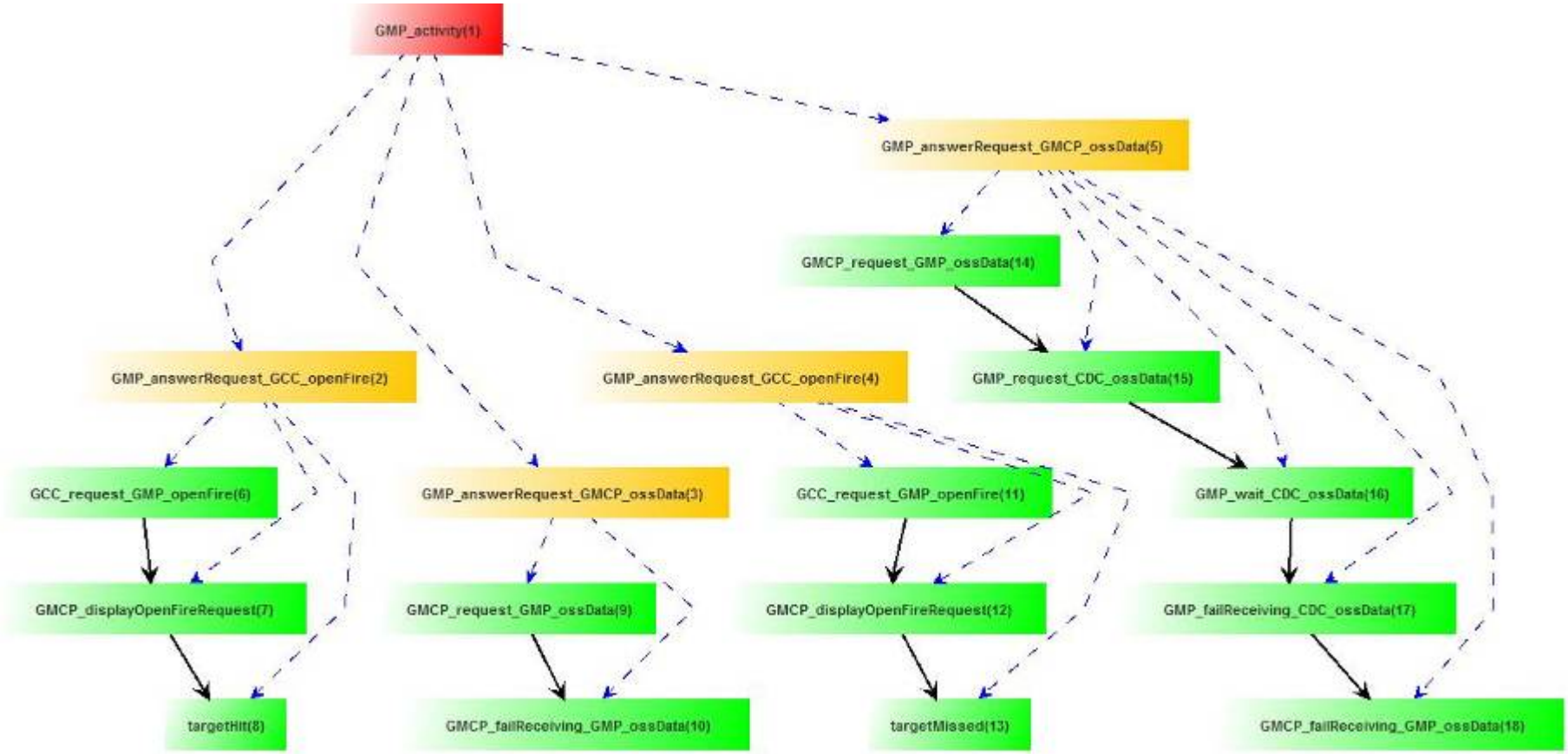


Figure 25: Scenario Generated from MP Schema: GMP_activity Scenario #96

ROOT CDC_activity - the activity of Optical Sight System MK 46 Mod 1-Control Display Console MK 132 Mod 0 [CDC]-The operator console used to control the MK46 Optical Sight

- CDC_answerRequest_GMP_ossData - CDC answers the request from GMP for oss data (waits for GMP to request oss data and provides it)
  - **CDC_request_EOD_ossData** - CDC request EOD oss data (thermal and daylight)
  - **CDC_wait_EOD_ossData** - CDC waits for oss data from EOD
  - **CDC_failReceiving_EOD_ossData** - CDC does not receive oss data from EOD
  - **CDC_receive_EOD_ossData** - CDC receives oss data from EOD

ROOT CDC_activity Model Results:

Scope: 4

Total Scenarios: 121

Processing Time: 0.08 Seconds

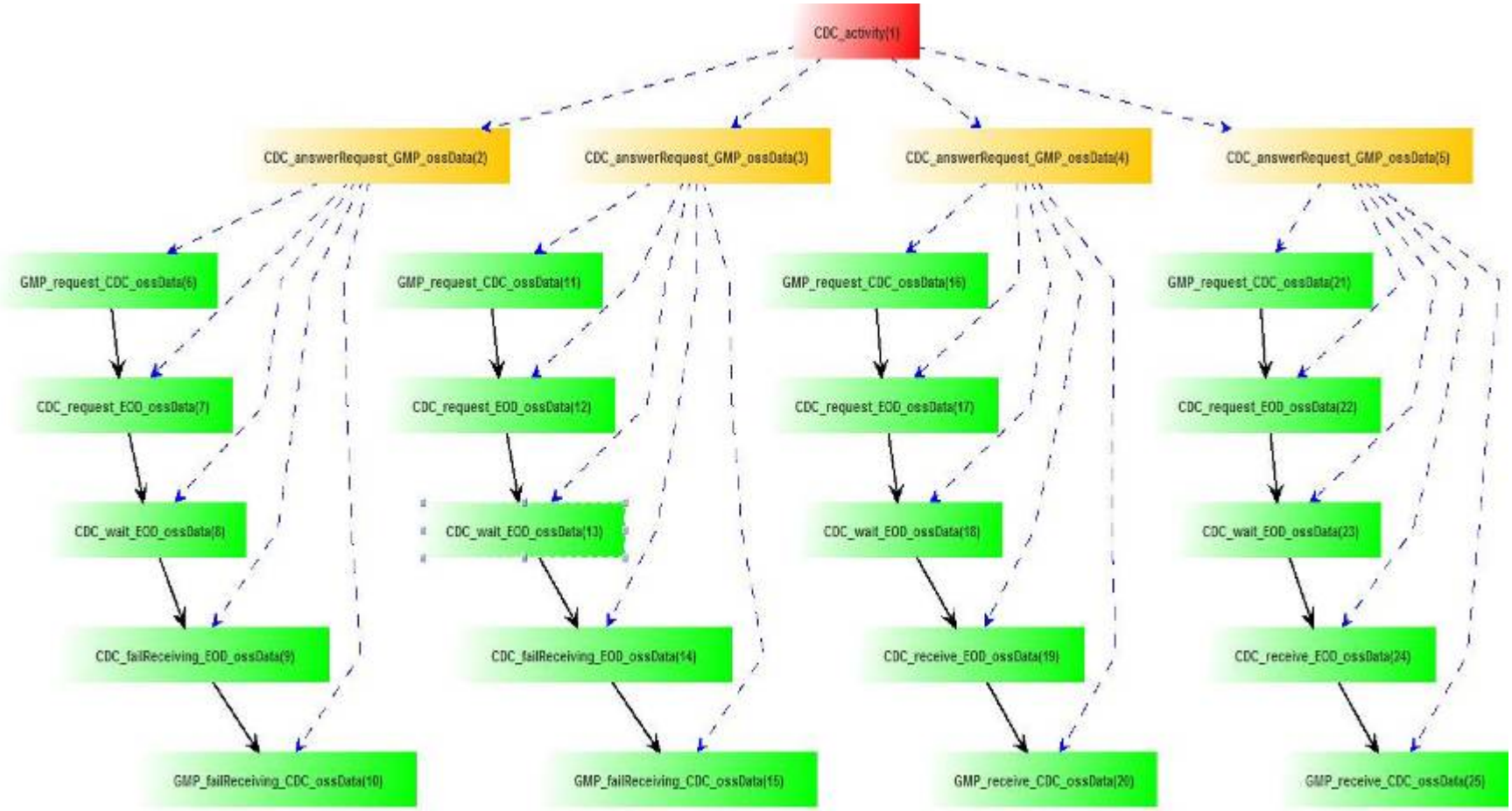


Figure 26: Scenario Generated from MP Schema: CDC_activity Scenario #85

ROOT EOD_activity-the activity of Optical Sight System MK 46 Mod 1-Electro-Optic Director MK 85 Mod 1 [EOD]-The Optical Sight director system (installed above the bridge) that rotates and elevates per operator's commands. The TV, IR, and laser rangefinder sensors are installed on the director, which points them in the right position

- EOD_answerRequest_CDC_ossData - EOD answers the request from CDC of oss data (waits for CDC to request oss data and provides it)
    - **EOD_requestDaylightSensorData** - EOD requests data from daylight sensor
    - **EOD_failGettingDaylightSensorData** - EOD fails to get data from daylight sensor
    - **EOD_receiveDaylightSensorData** - EOD receives data from daylight sensor
    - **EOD_requestThermalSensorData** - EOD requests data from thermal sensor
    - **EOD_failGettingThermalSensorData** - EOD fails to get data from thermal sensor
    - **EOD_receiveThermalSensorData** - EOD receives data from thermal sensor

ROOT EOD_activity Model Results:

Scope: 5

Total Scenarios: 1365

Processing Time: 1.91 Seconds

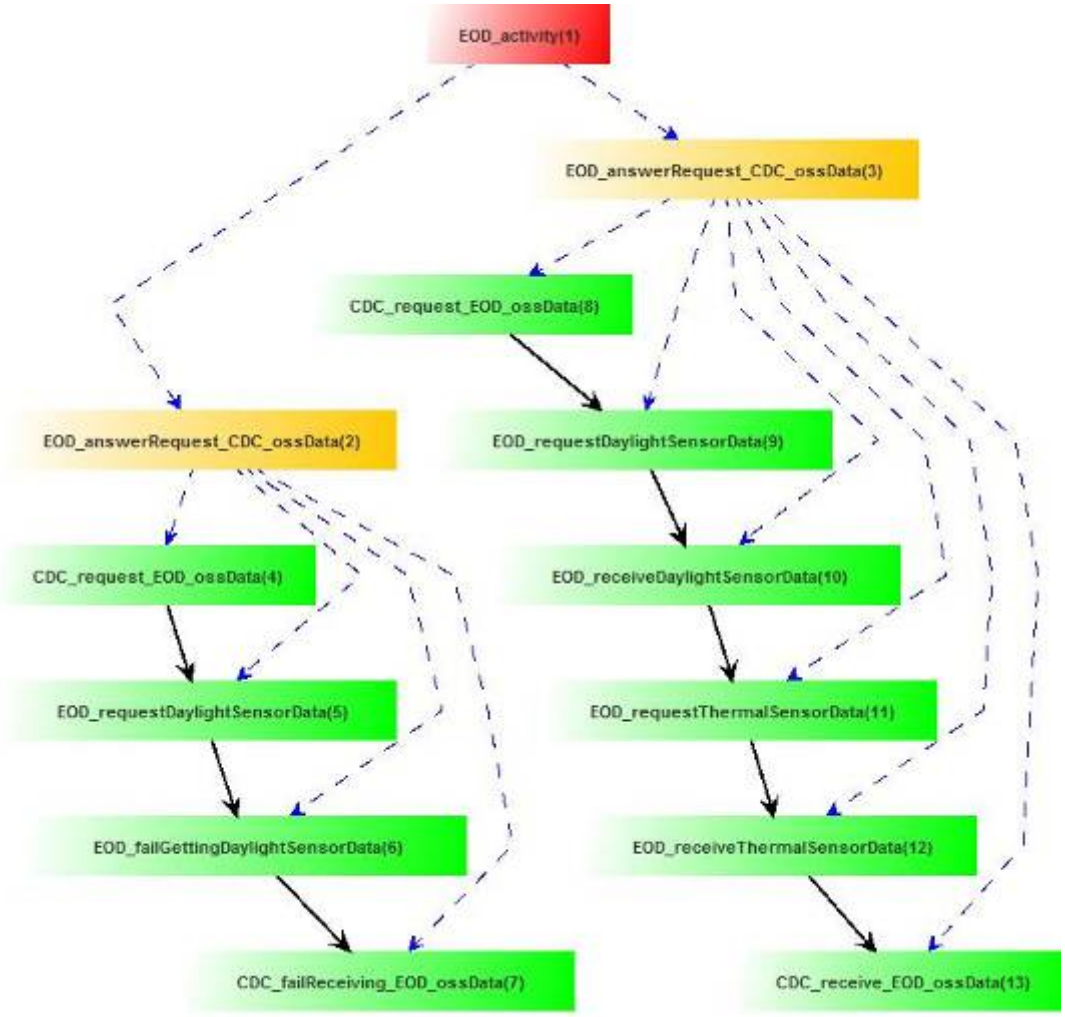


Figure 27: Scenario Generated from MP Schema: EOD_activity Scenario #13

ROOT GMCP_activity-the activity of Gun Mount Control Panel MK 437 Mod 1 [GMCP]-Backup Operator's console installed below the gun mount. It is used in case the main ADS console in the combat information center goes down

- GMCP_answerFireRequest - GMCP answers a fire request when displayed on screen (waits for a fire request to be displayed on the screen and it starts fire procedures)
  - **GMCP_displayOpenFireRequest** - GMCP displays an open fire request on screen
  - **GMCP_openFireFailed** - GMCP fails to open fire

- **GMCP_request_GMP_ossData** - GMCP requests optical sight system target data from GMP
- **GMCP_wait_GMP_ossData** - GMCP waits for optical sight system data from GMP
- **GMCP_failReceiving_GMP_ossData** - GMCP does not receive oss target data from GMP
- **GMCP_receive_GMP_ossData** - GMCP receives optical sight system data
- **GMCP_send_GM_openFireCommand** - GMCP sends GM an open fire command
- **GMCP_wait_GM_openFireCommand** - GMCP waits for GM to open fire

ROOT GMCP_activity Model Results:

Scope: 5

Total Scenarios: 1365

Processing Time: 1.91 Seconds

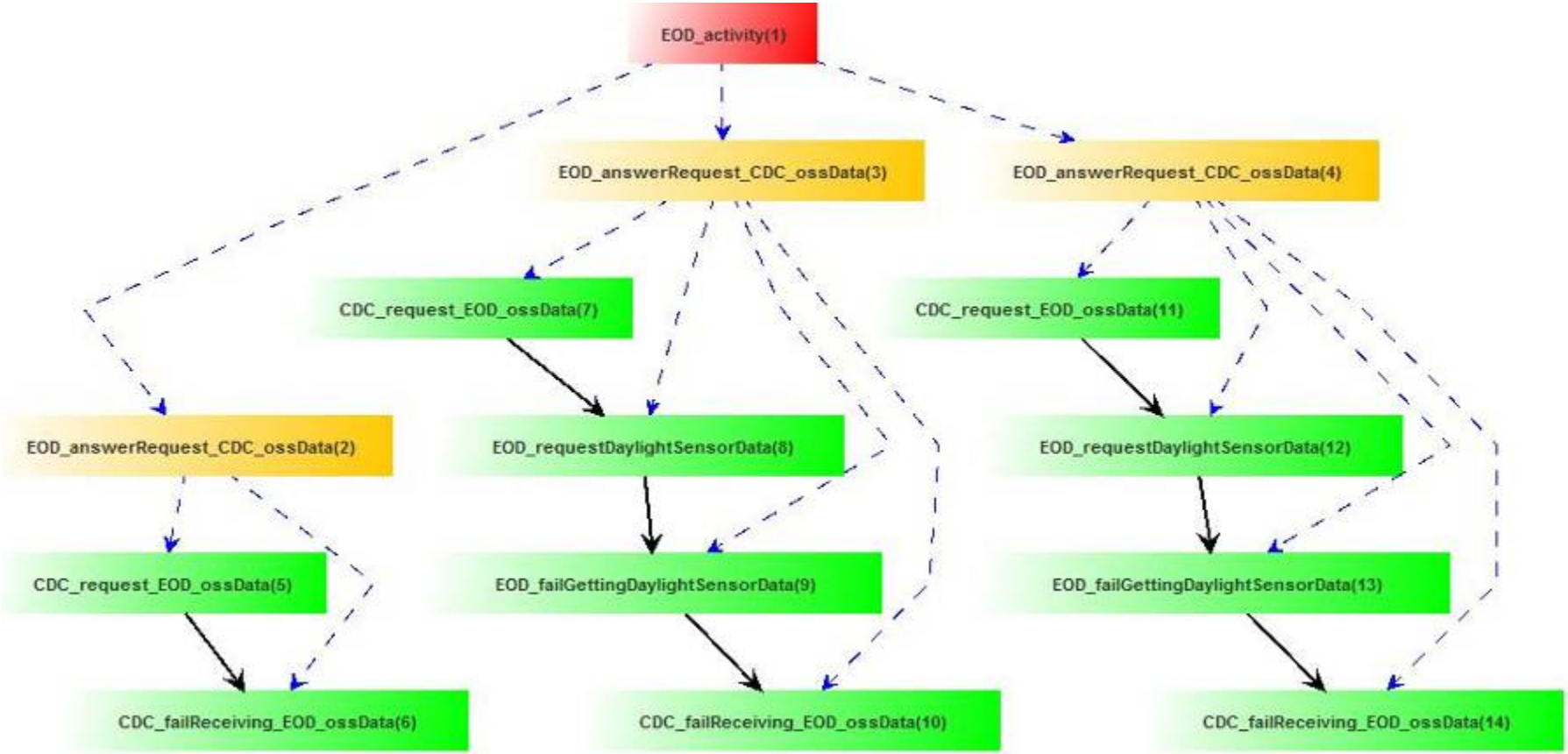


Figure 28: Scenario Generated from MP Schema: GMCP_activity Scenario #27

ROOT GM_activity-the activity of Gun Mount EX 45 Mod 4 [GM]-The 5” gun mount. Holds 20 rounds in the drum and fires 18-20 rounds per minute.

- GM_answer_GMCP_openFireCommand - waits for GMCP to send an open fire command and it opens fire
  - **GM_launchMissile** - GM launches a missile
  - **GM_waitForMissileToHit** - GM waits for the missile to hit the enemy target
  - **targetHit** - target is hit
  - **targetMissed** - target is missed

ROOT GM_activity Model Results:

Scope: 9

Total Scenarios: 1023

Processing Time: 1.97 Seconds

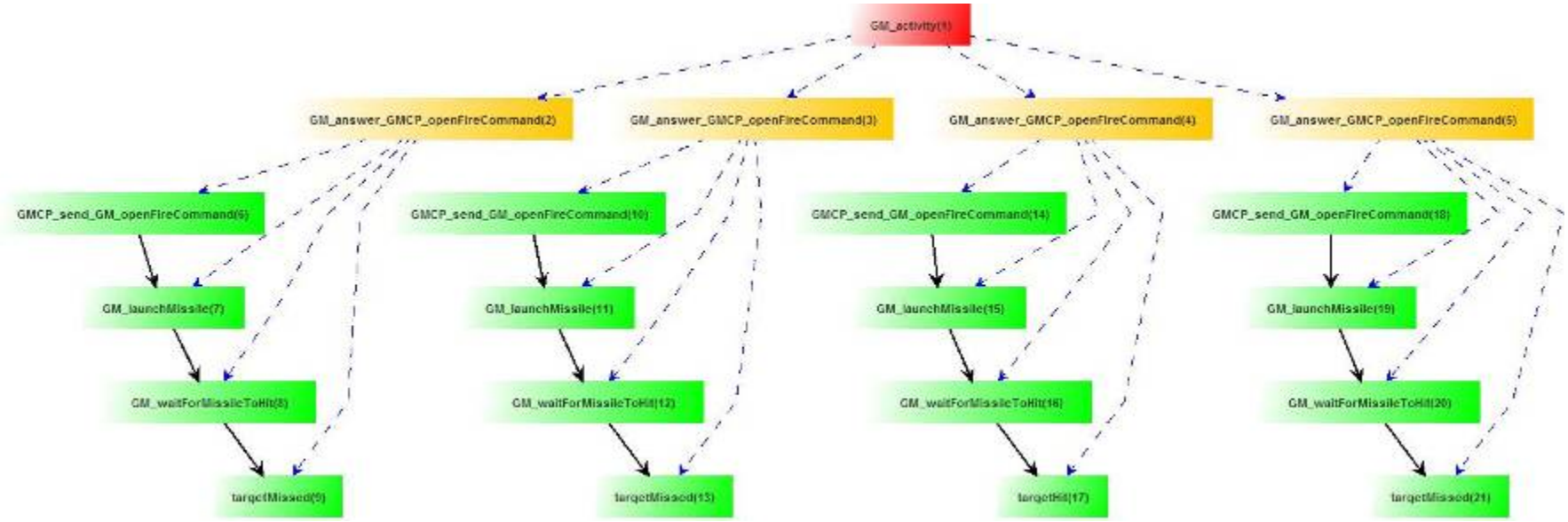


Figure 29: Scenario Generated from MP Schema: GM_activity Scenario #29

ROOT R3D_activity-the activity of AN/SPY-1D [R3D]-3-D Air Defense and Surface Search Phased Array Radar

- R3D_setTarget - R3D sets target on radar (it waits for GCC to request and returns additional information about target)
  - **R3D_targetNotSet** - R3D manages to set target
  - **R3D_targetSet** - R3D fails to set target

Code:

ROOT R3D_activity: {*R3D_setTarget*}; *// R3D sets a target*

R3D_targetNotSet, R3D_targetSet;

R3D_setTarget: (*// R3D sets target and waits*

GCC_request_R3D_setTarget *// waits for GCC response*

(R3D_targetNotSet *// R3D manages to set target*

| R3D_targetSet));*// R3D fails to set target*

R3D_activity, GCC_activity SHARE ALL GCC_request_R3D_setTarget,

---

ROOT R3D_activity Model Results:

Scope: 9

Total Scenarios: 1023

Processing Time: 9.57 Seconds

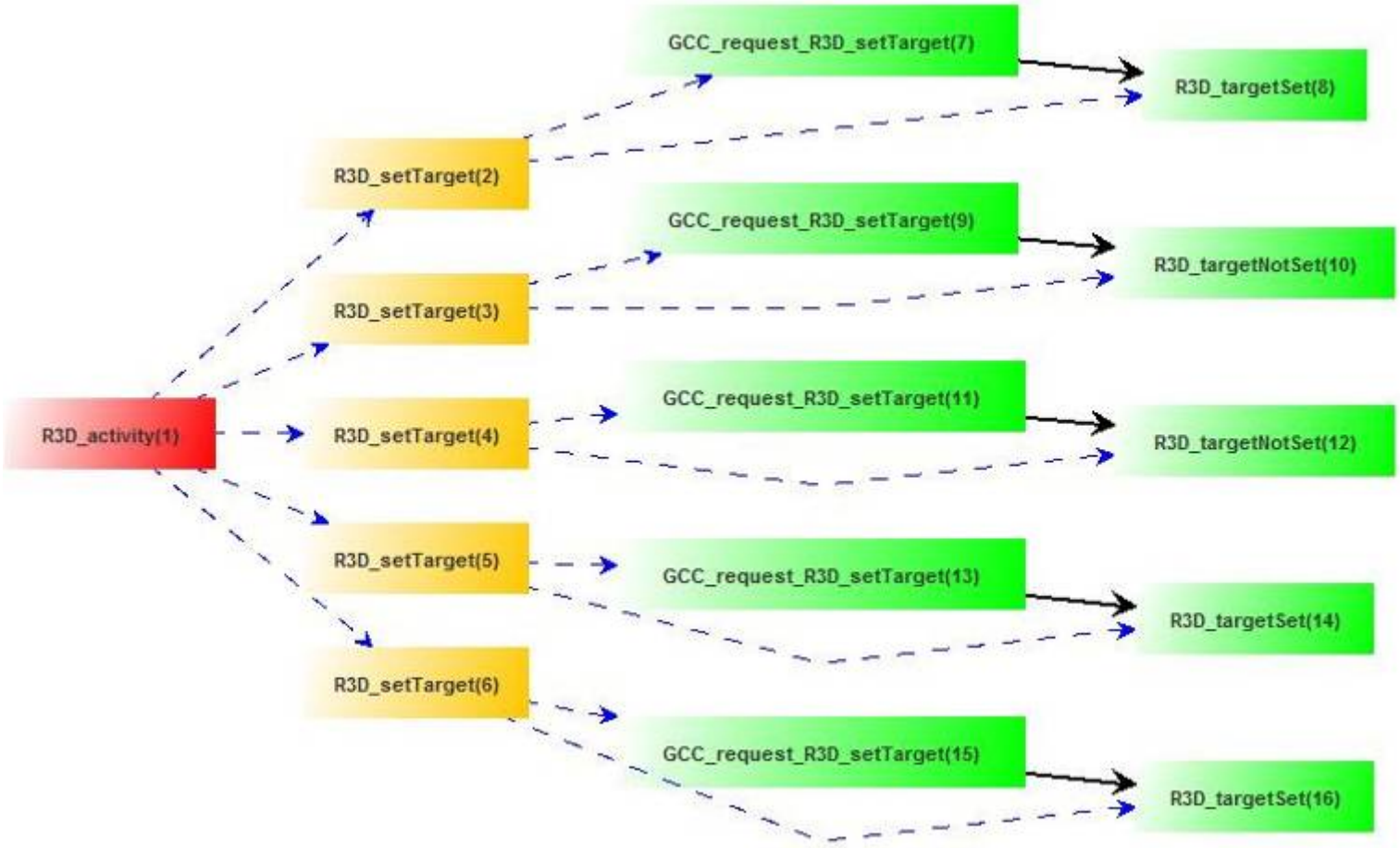


Figure 30: Scenario Generated from MP Schema: R3D_activity Scenario #53

a. Explanation of Event Attributes

There are two types of attributes: static and dynamic. Static attributes are values that are set at the beginning of a model and do not change. Dynamic attributes have a value that may change in different parts of the scenario. The Eagle6 prototype uses static attributes that enable query language. Dynamic attributes are reserved for future research. For more details on Dynamic attributes, see (Auguston & Whitcomb, System architecture specification based on behavior models, 2010).

Environmental Behavior

Events and schemas are used to model environmental behavior in the same way we model system behavior. Attributes are properties of events. For example, the following attribute "Req_Num_Man_Approv_For_Cmd" is used to measure the total number of manual approvals required to execute an event. The environmental behavior is the manual approval, such that the system and the environment both share the event. The following attributes are used within the gun weapon system model:

- Max_Watts - A numeric value of the amount of Watts required to execute the event. The default value is 0.
- Network_Bandwidth_Req_MB - Amount of network bandwidth required to transmit the event response. The measurement for this attribute is MB, and the default value is 0.
- Total_Processing_Time_Sec - Total time required for the event to elapse. The measurement for this attribute is seconds, and the default value is 0.
- Req_Num_Man_Approv_For_Cmd - Total number of manual approvals required to execute an event. This attribute is used to

find scenarios where the number of manual approvals required to process an OpenFire command exceeds the acceptable limit. The default value is zero.

Table 6: Gun Weapon System Model Events and Attributes

R2D_displayNewTarget: <Max_Watts=90, Network_Bandwidth_Req_MB=1.5, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1> ;
CD_request_GCC_setTarget: <Max_Watts=120, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1> ;
CD_wait_GCC_setTarget: <Max_Watts=10, Network_Bandwidth_Req_MB=0.1, Req_Num_Man_Approv_For_Cmd=1>;
GCC_request_R3D_setTarget: <Max_Watts=100, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1> ;
GCC_wait_R3D_setTarget: <Max_Watts=8, Network_Bandwidth_Req_MB=0.1, Req_Num_Man_Approv_For_Cmd=1>;
R3D_targetSet: <Max_Watts=120, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1> ;
GCC_targetSet: <Max_Watts=120, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1> ;
CD_followTarget: <Max_Watts=160, Network_Bandwidth_Req_MB=4.0, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=1> ;
CD_request_GCC_openFire: <Max_Watts=60, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=2> ;
CD_wait_GCC_openFire: <Max_Watts=5, Network_Bandwidth_Req_MB=0.2, Req_Num_Man_Approv_For_Cmd=2>;
GCC_request_GMP_openFire: <Max_Watts=60, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=2> ;
GCC_wait_GMP_openFire: <Max_Watts=5, Network_Bandwidth_Req_MB=0.3, Req_Num_Man_Approv_For_Cmd=2>;
GMCP_displayOpenFireRequest: <Max_Watts=140, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=0> ;
GMCP_request_GMP_ossData: <Max_Watts=100, Network_Bandwidth_Req_MB=2.5, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1> ;
GMP_request_CDC_ossData: <Max_Watts=100, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1> ;
GMP_wait_CDC_ossData: <Max_Watts=50, Network_Bandwidth_Req_MB=0.5, Req_Num_Man_Approv_For_Cmd=1>;
CDC_request_EOD_ossData: <Max_Watts=110, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1> ;
CDC_wait_EOD_ossData: <Max_Watts=100, Network_Bandwidth_Req_MB=1, Req_Num_Man_Approv_For_Cmd=1>;
EOD_requestDaylightSensorData: <Max_Watts=80, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1> ;
EOD_receiveDaylightSensorData: <Max_Watts=120, Network_Bandwidth_Req_MB=3.0, Total_Processing_Time_Sec=2.5, Req_Num_Man_Approv_For_Cmd=1> ;

EOD_requestThermalSensorData: <Max_Watts=80, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1> ;
EOD_receiveThermalSensorData: <Max_Watts=120, Network_Bandwidth_Req_MB=3.0, Total_Processing_Time_Sec=2.5, Req_Num_Man_Approv_For_Cmd=1> ;
CDC_receive_EOD_ossData: <Max_Watts=150, Network_Bandwidth_Req_MB=3.5, Total_Processing_Time_Sec=3.0, Req_Num_Man_Approv_For_Cmd=1> ;
GMP_receive_CDC_ossData: <Max_Watts=120, Network_Bandwidth_Req_MB=2.5, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1> ;
GMCP_receive_GMP_ossData: <Max_Watts=120, Network_Bandwidth_Req_MB=2.5, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1> ;
GMCP_wait_GMP_ossData: <Max_Watts=10, Network_Bandwidth_Req_MB=0.5, Req_Num_Man_Approv_For_Cmd=1>;
GMCP_send_GM_openFireCommand: <Max_Watts=100, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=2> ;
GMCP_wait_GM_openFireCommand: <Max_Watts=10, Network_Bandwidth_Req_MB=0.5, Req_Num_Man_Approv_For_Cmd=2>;
GM_launchMissile: <Max_Watts=250, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=2.5, Req_Num_Man_Approv_For_Cmd=2> ;
GM_waitForMissileToHit: <Max_Watts=50, Network_Bandwidth_Req_MB=0.5, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=0> ;

4. Testing Architectural Design Via Formal Queries

Identifying unintended system behavior is paramount when executing a software system safety assessment.

The concept “Chain” is defined as a set of events with the property that any two events from the chain (x and y) have a PRECEDES relationship between them (either x PRECEDES y, or y PRECEDES x). The set that contains all Chains of scenario s is described as Chain(s). Given a scenario s, we define A as a chain of s as:

$$A = \{x \mid x \in \text{Events}(s)\}, \text{ with the following property: } \forall\ x,y \in A,$$
$$\text{we have } (x \text{ PRECEDES } y) \text{ or } (y \text{ PRECEDES } x).$$

The formal design of these models creates a framework for system behavior properties to be expressed as computations over event traces. Eagle6 uses the MP framework and therefore supports extracting different views from the model, and verification of behavior properties within a given scope. Advantages of this approach compared with the common simulation tools are as follows:

- Means to write assertions about the system behavior and tools to verify those assertions.
- Exhaustive search through all possible scenarios (up to the scope limit).
- The support for verifiable refinement of the architecture model, up to design and implementation models.
- Integration of the architecture models with environment models for defining typical scenarios (use cases) and verifying system’s behavior for those scenarios.

The application of the gun weapon system model has the following two major functions: (1) testing the gun weapon system architectural design; and (2) generating random scenarios according to predefined probabilities with the purpose of getting different types of estimates.

a. Testing Architectural Design Via Formal Queries

The result of executing an MP model is a set of valid event traces (scenarios):

Figure 31: MP Model Scenario Generation Process

The following query language represents how the user obtains a set of scenarios by constructing dynamic queries via the Eagle6 user interface:

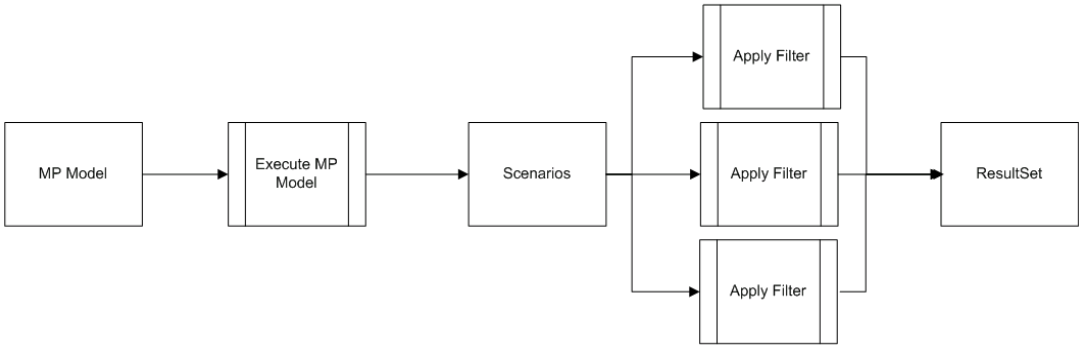


Figure 32: Query Building Process

Eagle6 has a graphical interface that enables the user to create queries. The graphical interface has three types of queries available in parameterized form as macro commands that can be used individually, or combined for a more refined ResultSet:

b. Macro Commands

Query type 1: EventCount

EventCount is used to return only scenarios that have a min/max number of total events within the scenario. The EventCount Macro Command has the following structure:

EventCount(EventType, Operator, Value)

Parameters required:

- EventType – a valid event type from the MP model
- Operator – one of the following: $\leq, <, =, >, \geq$
- Value – numerical value

Notation:

$$\mathbf{ResultSet} = \{s \in \mathbf{Scenarios} \mid \{x: \mathbf{EventType} \in s\} \mid \mathbf{Operator\ Value}\}$$

Example:

The following macro command returns all scenarios that have $\geq$ 1 events of type GCC_openFireFailed:

ResultSet = EventCount(GCC_openFireFailed, >=, 1)

Query type 2: SliceSum

SliceSum is used to return only scenarios that have a min/max number of total events that happen in parallel within the scenario. The SliceSum Macro Command has the following structure:

SliceSum(AttributeName, Operator, Value)

Parameters required:

- AttributeName – a valid attribute name from the MP model
- Operator – one of the following: <, <=, =, >, >=
- Value – numerical value

Notation:

MaxSliceSum:

$$\mathbf{ResultSet} = \{s \in \mathbf{Scenarios} \mid \exists\, x: \mathbf{Slice} \in s \ \mathbf{such\ that\ SUM}\ [e\ \mathbf{from}\ x\ e.\mathbf{AttributeName}]\mathbf{Operator\ Value}\}$$

Example:

The following query returns all scenarios that have at least one Slice of events where the attribute MaxWatts aggregate sum ≥ 220:

SliceSum(MaxWatts, >=, 220)

Query type 3: ChainSum

ChainSum is used to return only scenarios with events that happen in sequence and also have attribute properties that meet the query definition. The ChainSum Macro Command has the following structure:

ChainSum(AttributeName, Operator, Value)

Parameters required:

- AttributeName – a valid attribute name from the MP model
- Operator – one of the following: $\leq, <, =, >, \geq$
- Value – numerical value
- Sum - the total sum of the Attribute values found in the ResultSet

Notation:

MaxChainSum:

$$\mathbf{ResultSet} = \{\mathbf{s} \in \mathbf{Scenarios} \mid \exists \mathbf{x}: \mathbf{Chain} \in \mathbf{s\ such\ that\ SUM}\ [\mathbf{e\ from\ x\ e.AttributeName}]\mathbf{Operator\ Value}\}$$

Example:

The following query returns all scenarios that have at least one chain of events that has an aggregate sum of the attribute Total_Processing_Time_Sec that is >= five:

ChainSum(Total_Processing_Time_Sec, >=, 5)

Query type 4: Combined Query

The Eagle6 interface has the ability to create combined queries. The ResultSet is generated from a combination of the predefined macro commands 1-3:

ResultSet = MacroCommand1 ∩ MacroCommand2

The intersection of sets MacroCommand1 and MacroCommand2 is the set of all elements of MacroCommand1 which are also elements of MacroCommand2.

Example:

The example represents a query that combines the Queries 1-3, and returns scenarios that meet the all of the queries' criteria:

Minimum of one scenario where the GCC_openFireFailed event Count >= 1 AND a minimum of one scenario where parallel events that have the attribute MaxWatts have an aggregate sum >=220.

ResultSet = {(*EventCount(GCC_openFireFailed, >=, 1))* ∩ *(SliceSum(MaxWatts, >=, 220))}*

F. IDENTIFYING POTENTIAL SOFTWARE SAFETY HAZARD STATES

When modeling the naval gun weapon system (Appendix A), we use both exhaustive and random scenario generation to evaluate software safety. Appendix B contains the Gun weapon system Assertion Library. The Exhaustive Search is the process of generating all possible scenarios from the MP model up to a given scope. The exhaustive search enables the user to find scenarios that produce counter-examples of assertions. The Random Approach is used to

generate estimates that are used for software safety assessment. Eagle6 generates random scenarios within scope to calculate statistical estimates.

Exhaustive Search

The exhaustive search method enables the user to input query criteria that customizes the result set returned by the software. Limiting result sets to important scenarios enables users to see only the data in which they are interested.

1. Modeling Demonstration: QUERY GWSMaxWatts

Hazard State: Find scenarios where the gun weapon system may require more Watts than the gun weapon system can produce.

Test Definition: Return all scenarios within scope that have at least one Slice that contains events that have the attribute MaxWatts, and the sum of the attribute MaxWatts is >= 220.

Macro Command:

ResultSet = SliceSum(MaxWatts, >=, 220)

Figure 33: QUERY GWSMaxWatts - Scenario Query

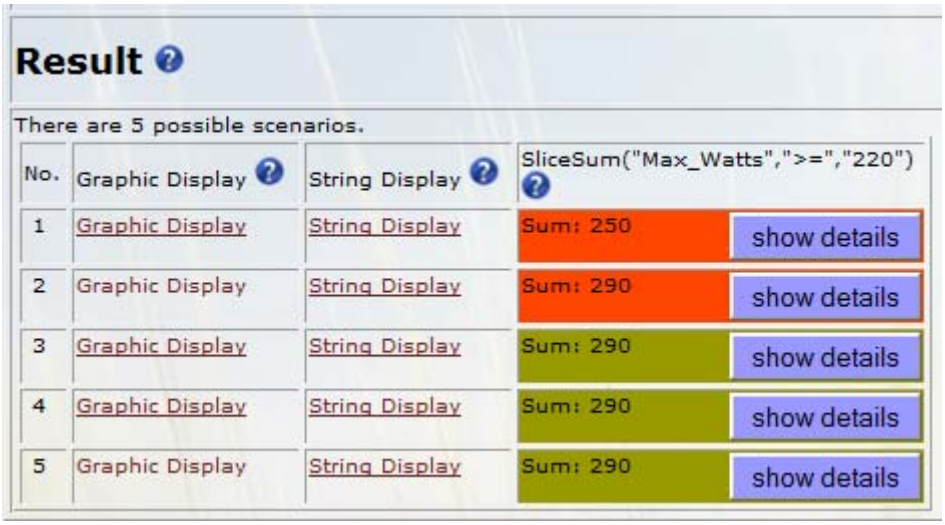


Figure 34: QUERY GWSMaxWatts - Results

The graphic contains the following information:

- Graphic Display - A hyperlink that is programmed to display the graphical image of the scenario in a new browser.

- Total Event Count - The total number of events that are included in the scenario.
- SliceSum(Max_Watts) for parallel events - The column is used to display scenario details that evaluate attributes. The "Show Details" button displays the individual events and their corresponding attribute values. The background color orange is used to alert the user that one or more events do not have an attribute value assigned. The textual output value for attribute values that are null is empty. The color green is used to alert the user that all events have assigned attribute values.

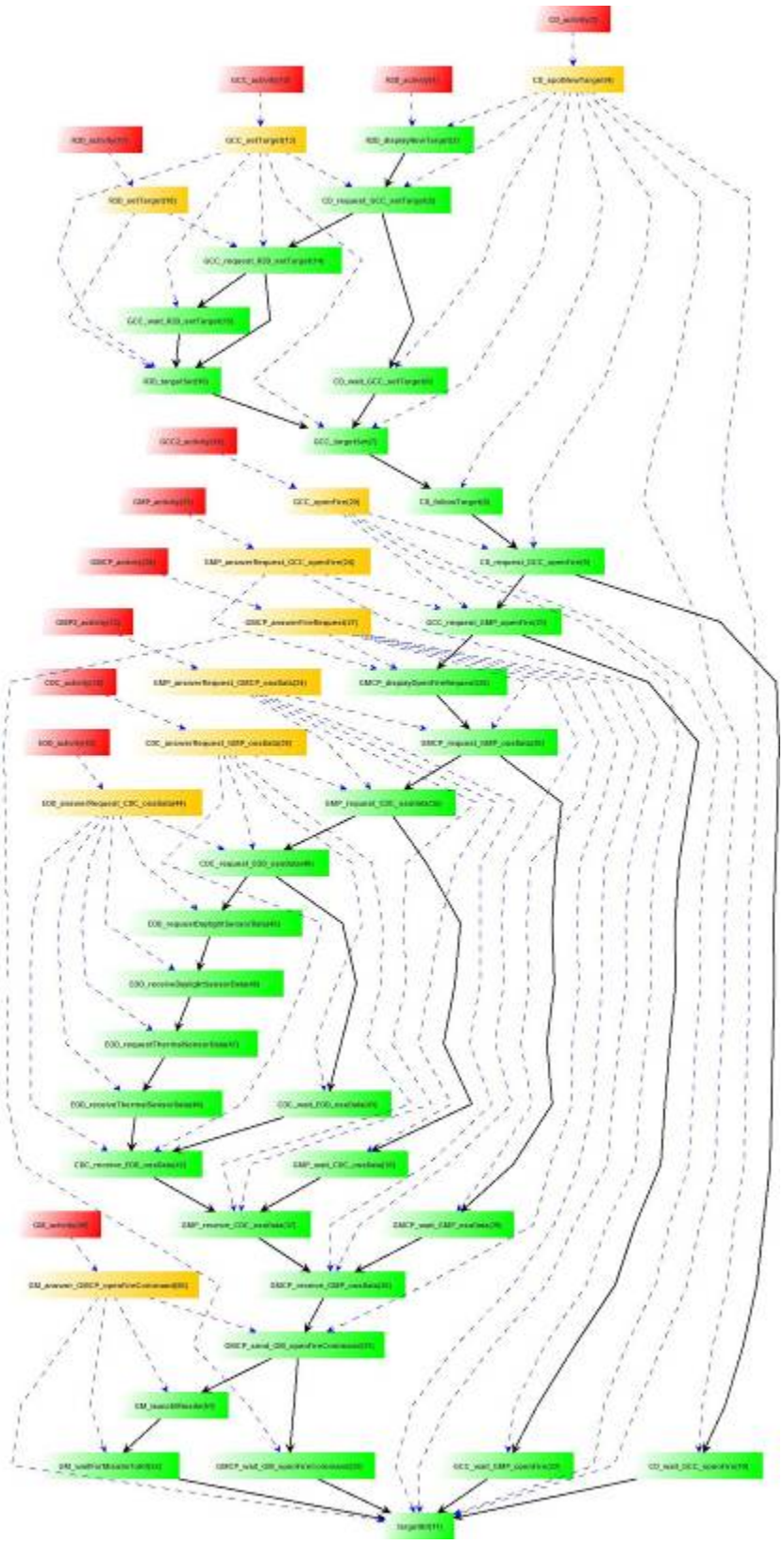


Figure 35: Scenario Generated from QUERY GWSMaxWatts - Graphical Display

The following graph represents a close-up view of the events that are identified in the SliceSum Query:

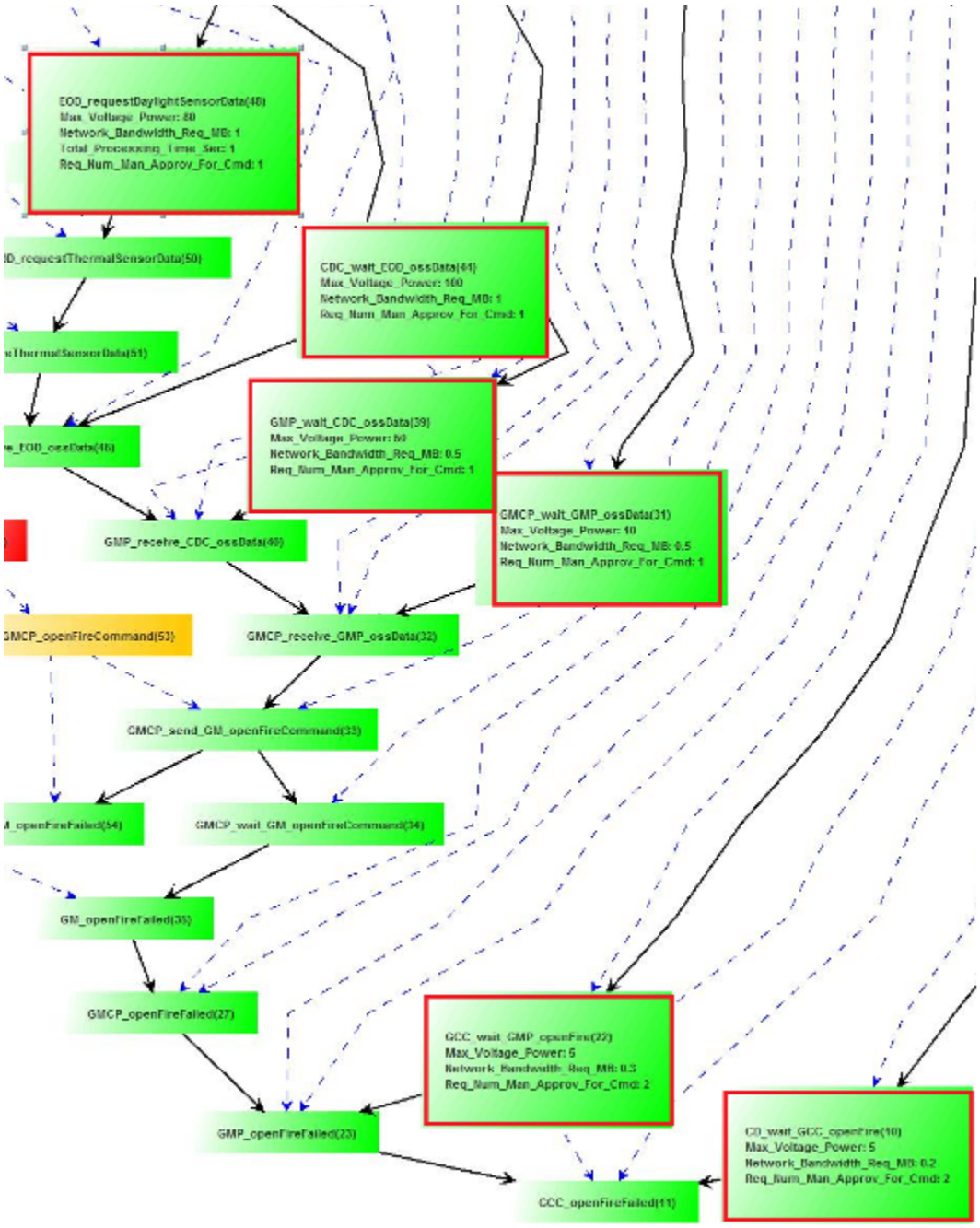


Figure 36: Scenario Generated from QUERY GWSMaxWatts - Zoom Slice View

Summary: The Software Safety Hazard State description ““Find scenarios where the gun weapon system may require more power than the gun weapon system can produce”” resulted in the test scenario containing the Macro Query SliceSum(MaxWatts, >=, 220). The query demonstration showed the query returned five possible scenarios where the gun weapon system could result in a Hazard State. The following is a list of events contained in the slice (that satisfied the query) from the scenario:

The slice from the scenario contains the following events:

Events (Sum of MaxWatts = 290)

- CD_wait_GCC_openFire(10).MaxWatts =5
- GCC_wait_GMP_openFire(22).MaxWatts =5
- GMCP_wait_GMP_ossData(29).MaxWatts =10
- GMP_wait_CDC_ossData(36).MaxWatts =50
- CDC_wait_EOD_ossData(41).MaxWatts =100
- EOD_receiveDaylightSensorData(46).MaxWatts =120

2. Modeling Demonstration: QUERY Network_Capacity_Check

Hazard State: Find scenarios where the gun weapon system may require more network bandwidth than the gun weapon system network can provide.

Test Description: Find a set of scenarios that have at least one slice with the following property: the sum of the attribute Max_Network_Bandwidth for all events that belong to that slice must be >= five .

Macro Command:

SliceSum(Max_Network_Bandwidth, >=, 5)

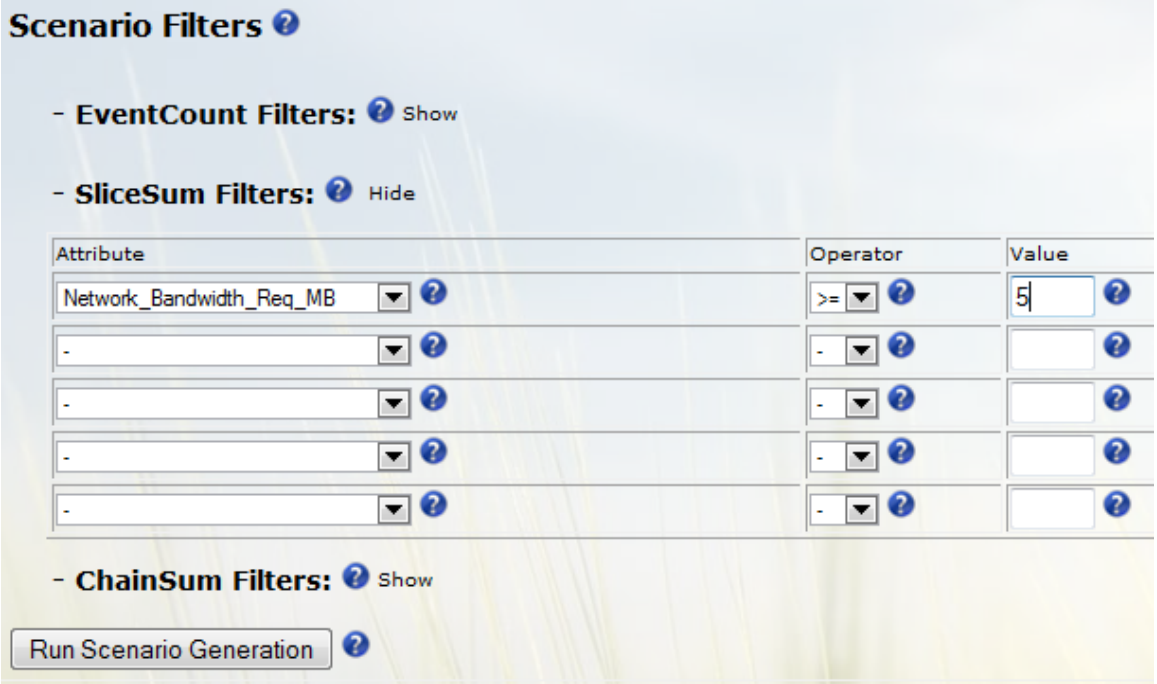


Figure 37: QUERY Network_Capacity_Check - Scenario Query

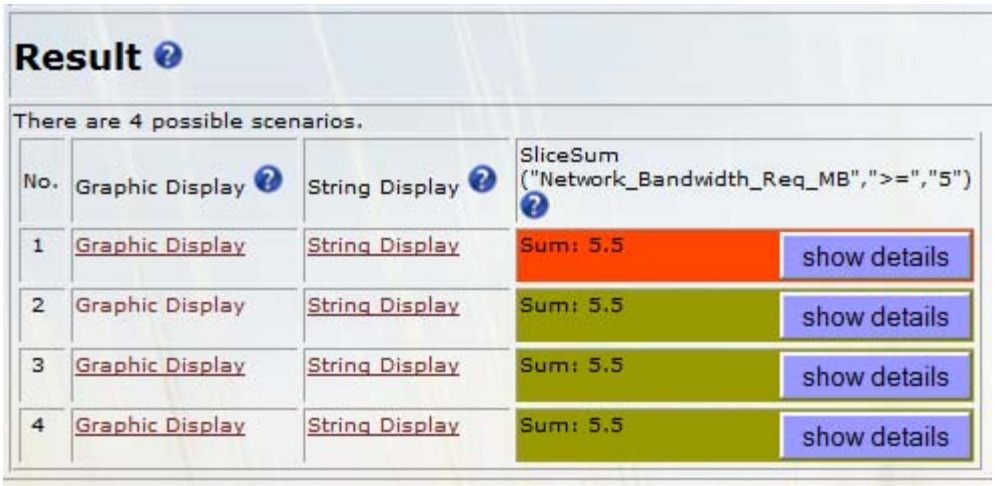


Figure 38: QUERY Network_Capacity_Check - Results

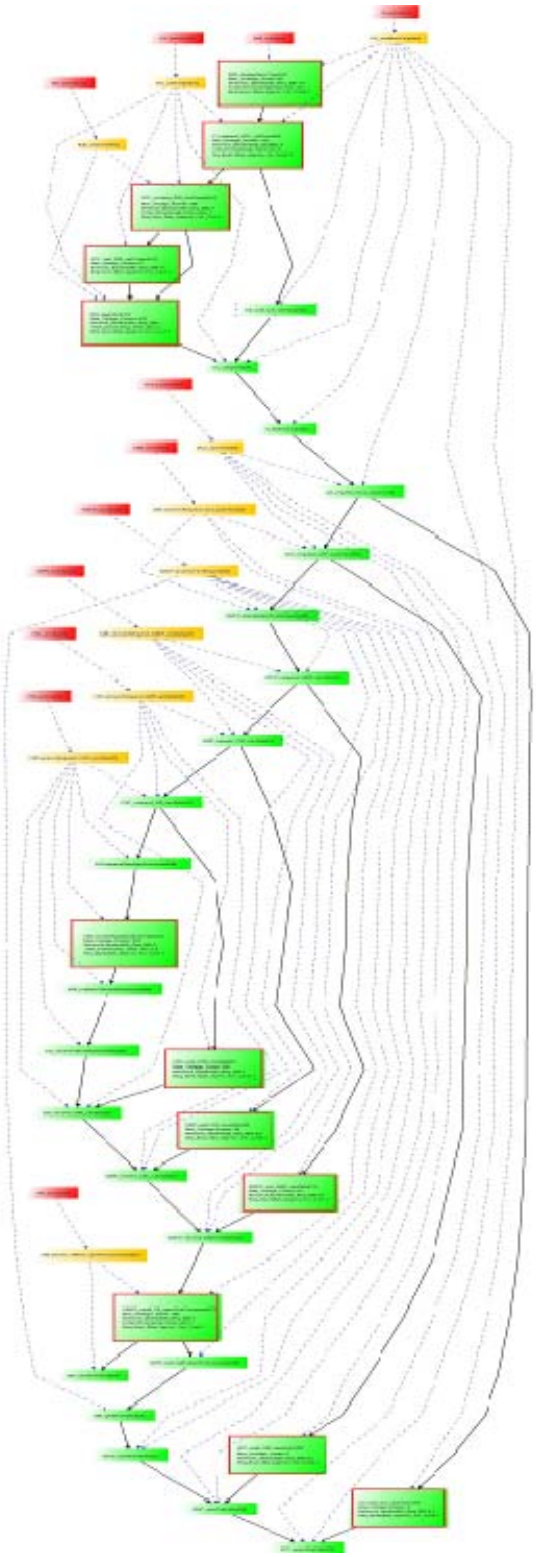


Figure 39: Scenario Generated from QUERY Network_Capacity_Check - Graphical Display

Summary: The Software Safety Hazard State description “Find scenarios where the gun weapon system may require more network bandwidth than the gun weapon system network can provide” resulted in the test scenario containing the Macro Query SliceSum(Max_Network_Bandwidth, >=, 5). The query demonstration showed the query returned four possible scenarios where the gun weapon system network could result in a Hazard State. The following is a list of events contained in the slice (that satisfied the query) from the scenario:

Events (Sum of Max_Network_Bandwidth_MB: 5.5)

- CD_wait_GCC_openFire(10).Max_Network_Bandwidth_MB =0.2
- GCC_wait_GMP_openFire(22).Max_Network_Bandwidth_MB =0.3
- GMCP_wait_GMP_ossData(31).Max_Network_Bandwidth_MB =0.5
- GMP_wait_CDC_ossData(39).Max_Network_Bandwidth_MB =0.5
- CDC_wait_EOD_ossData(44).Max_Network_Bandwidth_MB =1
- EOD_receiveDaylightSensorData(49).Max_Network_Bandwidth_MB =3

3. Model Demonstration: QUERY GCC_OpenFireFail

Hazard State: Find scenarios where the gun weapon system may experience the failure of a Gun Control Center Open Fire Command.

Test Definition: Find a set of possible hazard state scenarios where the CGG_openFire event happens at least once.

Macro Command:

ResultSet = EventCount(GCC_openFireFailed, >=, 1)

Figure 40: QUERY GCC_OpenFireFail - Scenario Query

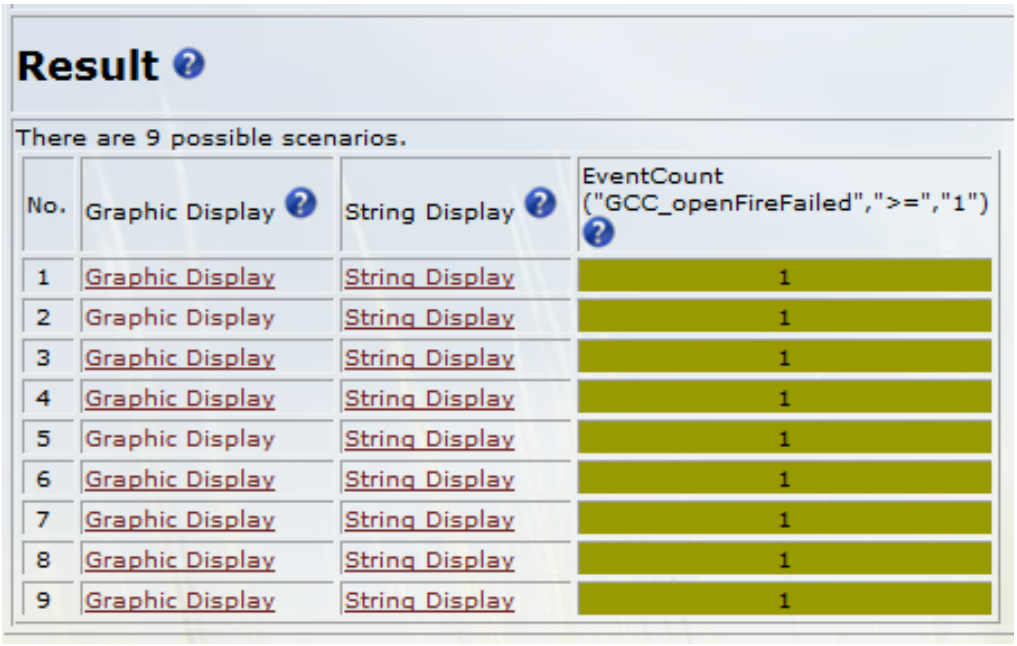


Figure 41: QUERY GCC_OpenFireFail - Results

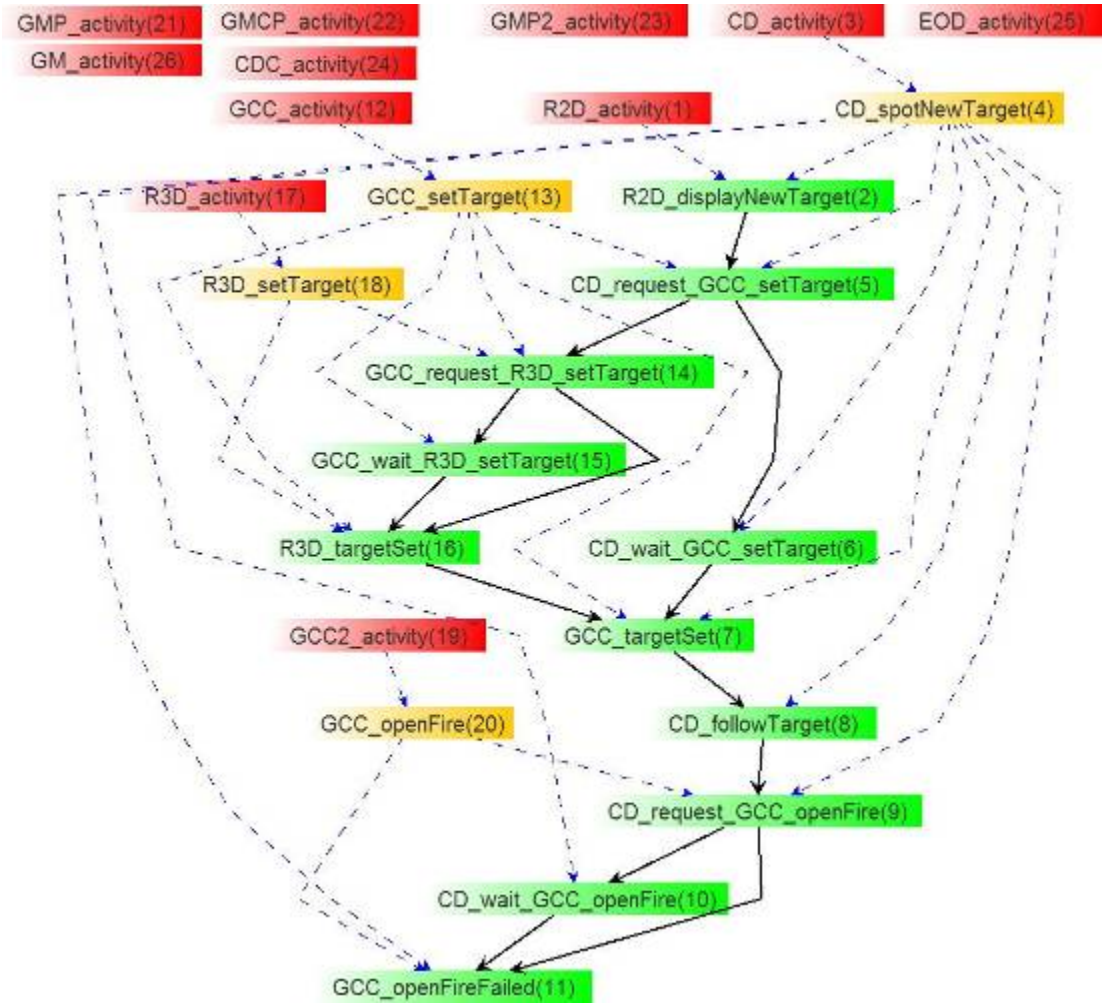


Figure 42: Scenario Generated from QUERY GCC_OpenFireFail - Graphical Display

Summary: The Software Safety Hazard State description “Find scenarios where the gun weapon system may experience the failure of a Gun Control Center Open Fire Command” resulted in the test scenario containing the Macro Query EventCount(GCC_openFireFailed, >=, 1). The query demonstration showed the query returned nine possible scenarios where the gun weapon system could result in a Hazard State.

4. Model Demonstration: QUERY Max_Manual_Approvals

Hazard State: Find scenarios where gun weapon system design may result in the gun weapon system requiring >= three manual approvals to execute an Open Fire Command.

Test Description: Find a set of scenarios that contain at least one GCC_openFire event, and also have at least one slice of events that have the Attribute Req_Num_Man_Approv_For_Cmd with a sum that is >= 3.

Macro Command:

ResultSet = {(EventCount(GCC_openFire, >=, 1))

(ChainSum(Req_Num_Man_Approv_For_Cmd,>=, 3))};

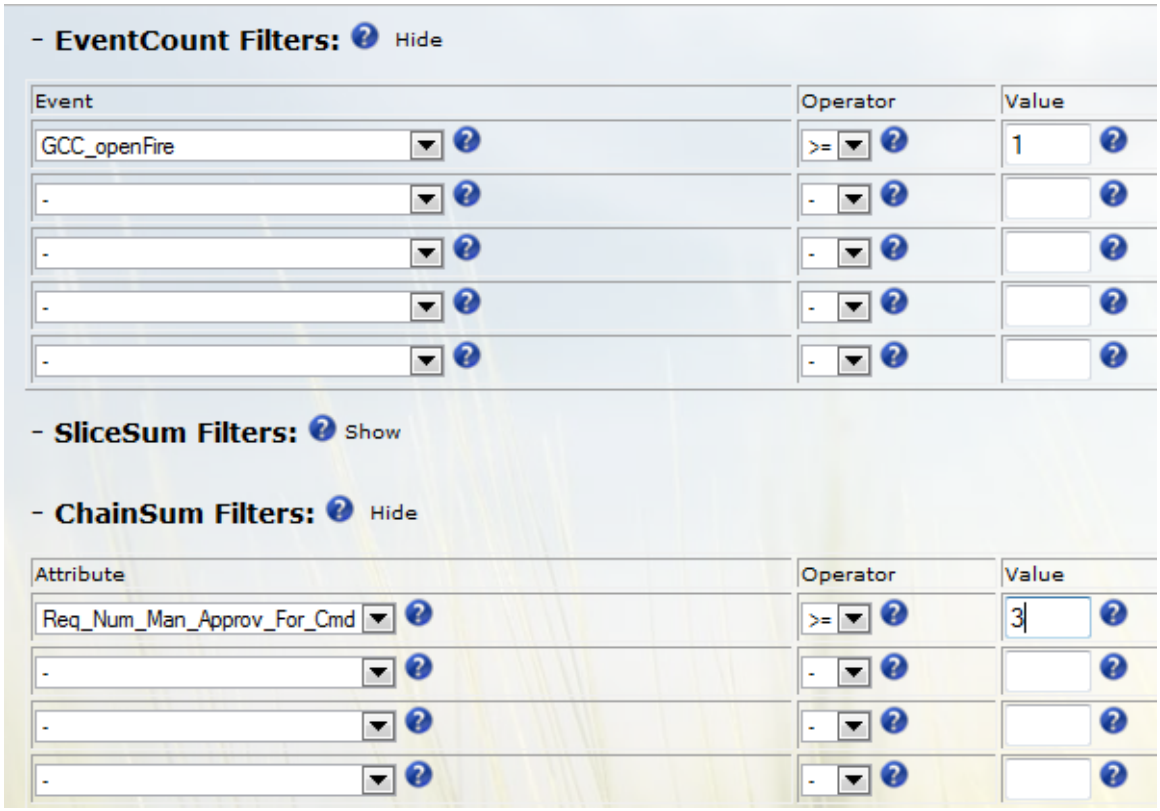


Figure 43: QUERY Max_Manual_Approvals - Scenario Query

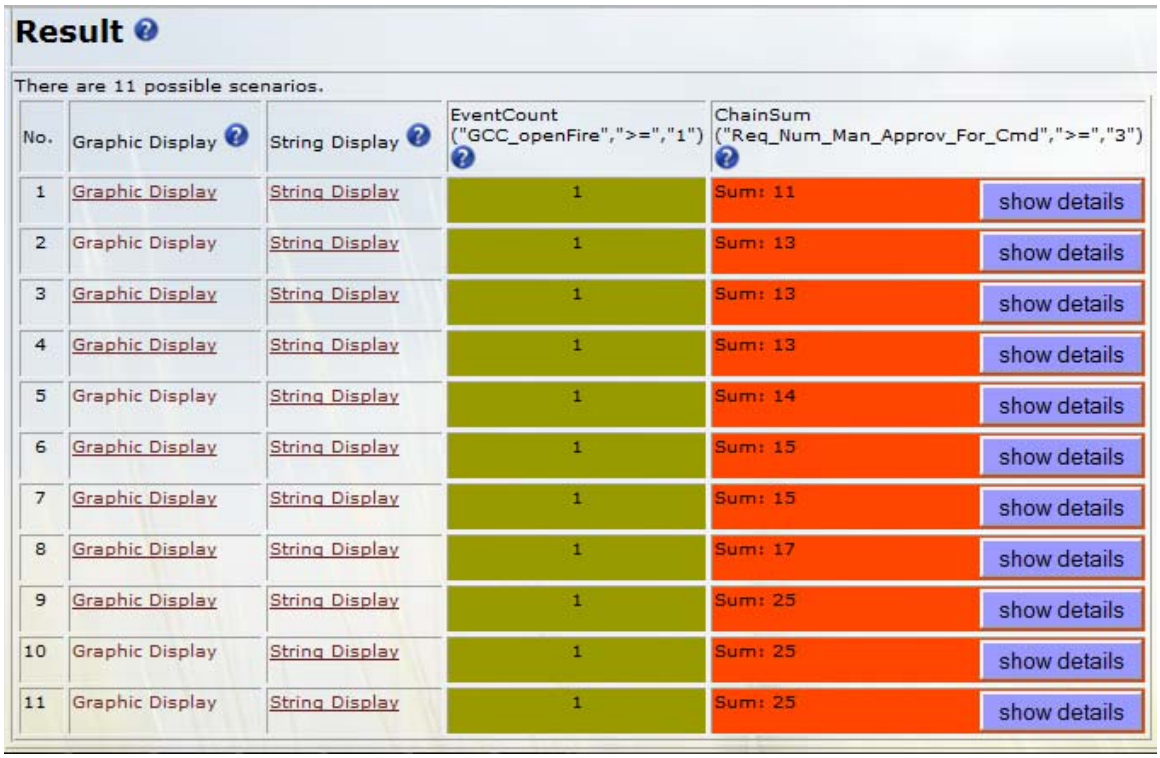


Figure 44: QUERY Max_Manual_Approvals - Results

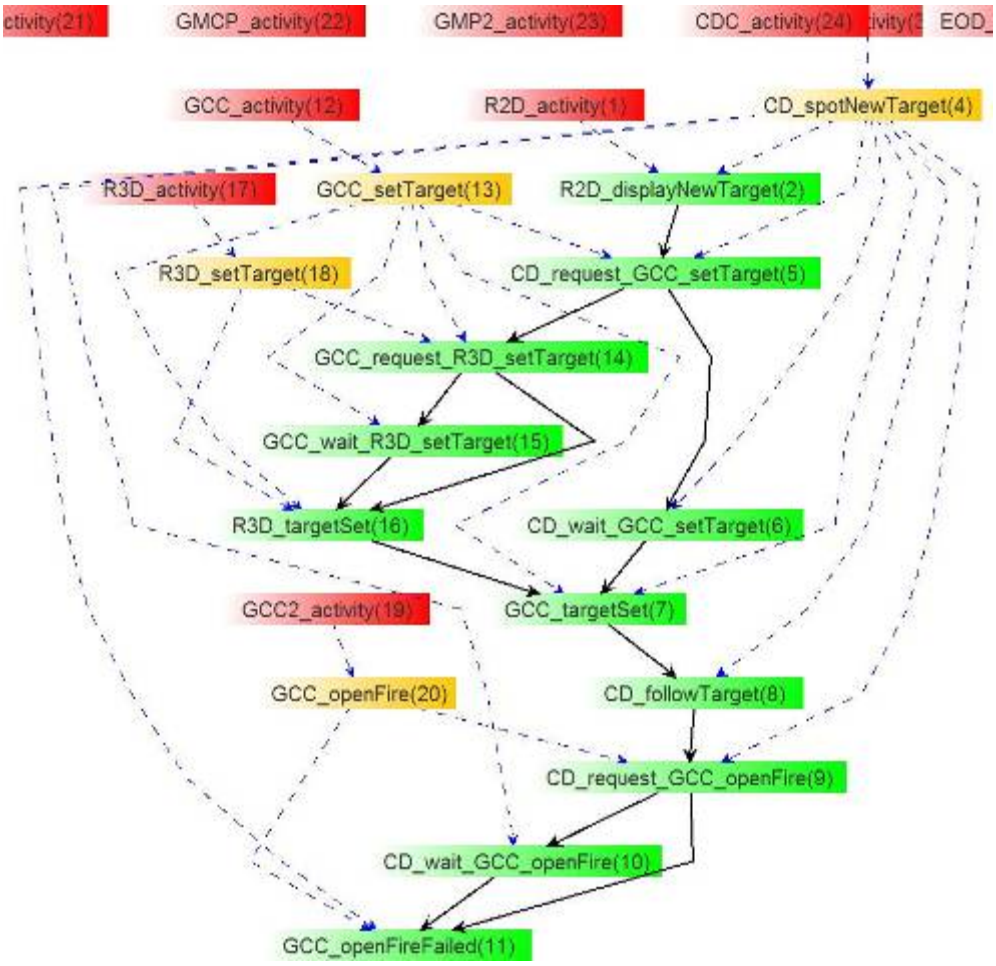


Figure 45: Scenario Generated from QUERY Max_Manual_Approvals - Graphical Display

Summary:

The Software Safety Hazard State description “Find scenarios where gun weapon system design may result in the gun weapon system requiring >= three manual approvals to execute an Open Fire Command” resulted in the test scenario containing the Macro Query:

{(EventCount(GCC_openFire, >=, 1))

(ChainSum(Req_Num_Man_Approv_For_Cmd,>=, 3))}.

The query demonstration showed the query returned ten possible scenarios where the gun weapon system could result in a Hazard State. The following is a list of events contained in the slice (that satisfied the query) from the scenario:

Events (Sum of Req_Num_Man_Approv_For_Cmd: 8)

- CD_wait_GCC_openFire(10).Req_Num_Man_Approv_For_Cmd =2
- GCC_wait_GMP_openFire(22).Req_Num_Man_Approv_For_Cmd =2
- GMCP_wait_GM_openFireCommand(32).Req_Num_Man_Approv_For_Cmd =2
- GM_launchMissile(51).Req_Num_Man_Approv_For_Cmd =2

5. Model Demonstration: QUERY GCC_ OpenFireFailed

Hazard State: Find scenarios where Gun Console Computer tries to execute an Open Fire Command and it ends with a system timeout.

Description: Find a set of scenarios that contain the GCC_openFire event, and also have at least one chain of events with a sum of the attribute Total_Processing_Time_Sec that is >= five .

Macro Command:

ResultSet = {(EventCount(GCC_openFire, >=, 1))

(ChainSum(Total_Processing_Time_Sec,>=, 5))};

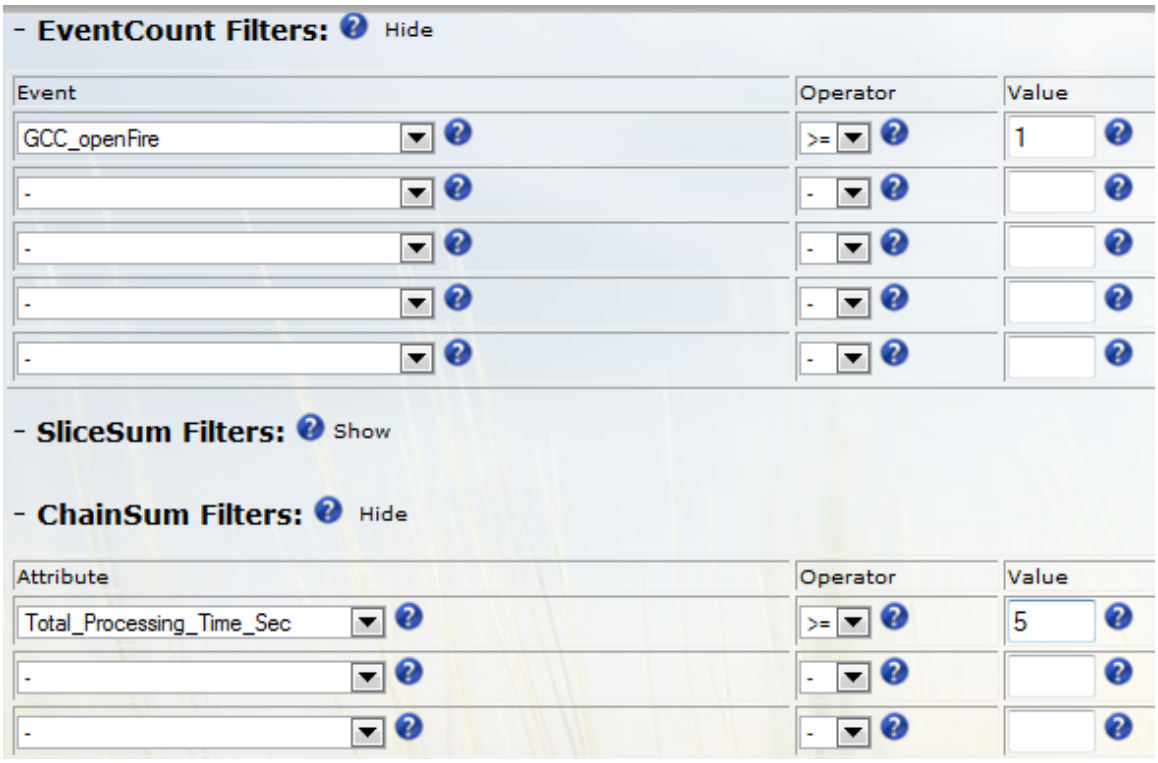


Figure 46: GCC_ OpenFire Total Processing Time - Scenario Query

Result

There are 11 possible scenarios.

No.	Graphic Display	String Display	EventCount ("GCC_openFire",">=","1")	ChainSum ("Total_Processing_Time_Sec",">=","5")	
1	Graphic Display	String Display	1	Sum: 8.5	show details
2	Graphic Display	String Display	1	Sum: 9	show details
3	Graphic Display	String Display	1	Sum: 10	show details
4	Graphic Display	String Display	1	Sum: 12	show details
5	Graphic Display	String Display	1	Sum: 13.5	show details
6	Graphic Display	String Display	1	Sum: 15	show details
7	Graphic Display	String Display	1	Sum: 16	show details
8	Graphic Display	String Display	1	Sum: 19.5	show details
9	Graphic Display	String Display	1	Sum: 29.5	show details
10	Graphic Display	String Display	1	Sum: 32.5	show details
11	Graphic Display	String Display	1	Sum: 32.5	show details

Figure 47: GCC_OpenFire Total Processing - Results

Figure 48: GCC_OpenFire Total Processing - Graphical Display

Summary: The Software Safety Hazard State description “Find scenarios where Gun Console Computer tries to execute an Open Fire Command and it ends with a system timeout” resulted in the test scenario containing the Macro Query:

{(EventCount(GCC_openFire, >=,))

(ChainSum(Total_Processing_Time_Sec,>=, 5))};

The query demonstration showed the query returned 11 possible scenarios where the gun weapon system could result in a Hazard State. The following is a list of events contained in the slice (that satisfied the query) from the scenario:

Events (SUM of Total_Processing_Time_Sec: 8.5 Seconds)

- R2D_displayNewTarget(2).Total_Processing_Time_Sec =1.5
- CD_request_GCC_setTarget(5).Total_Processing_Time_Sec =1
- GCC_targetSet(7).Total_Processing_Time_Sec =2
- CD_followTarget(8).Total_Processing_Time_Sec =0.5
- CD_request_GCC_openFire(9).Total_Processing_Time_Sec =0.5
- CD_wait_GCC_openFire(10).Total_Processing_Time_Sec = Attribute Value Not Assigned
- GCC_openFireFailed(11).Total_Processing_Time_Sec = Attribute Value Not Assigned
- GCC_request_R3D_setTarget(14).Total_Processing_Time_Sec =1
- GCC_wait_R3D_setTarget(15).Total_Processing_Time_Sec = Attribute Value Not Assigned
- R3D_targetSet(16).Total_Processing_Time_Sec =2

G. USING PROBABILITIES TO REFINE SYSTEM BEHAVIOR IN MP

Inserting event probabilities is designed to give the modeler a more refined capability of modeling actual system behavior. Introducing event probabilities may be used to estimate the probability of a Hazard State, as well as the probability of an Software Safety assertion.

Eagle6 uses the Monte Carlo method of approximating an expectation by the sample mean of a function of simulated random variables within the model scope

The following iterative scenario generation process, and random scenario generation process require a larger scope in order to generate statistical results that represent actual system behavior:

- Iterative Scenario Generation - A process that uses the Markov Chain theory such that the ResultSet consists of a finite number of states (scope) and some known probabilities p, where p_{ij} is the probability of moving from state i to state j. This approach enables the user to generate scenarios that produce counter-examples of assertions, as well as the probabilities of those assertions.
- Random Scenario Generation - Generates random scenarios within scope to calculate statistical estimates. The purpose of this functionality is to create estimates that are used for software safety assessments.

To determine probabilities of a scenario, event attributes are assigned a probability value:

(* <0-n/a_0,a_1,a_2,a_3 ... a_n> Radar_Target_Identified *)

The notation represents an a_0 probability that *Radar_Target_Identified* appears zero times, an a_1 probability that *Radar_Target_Identified* appears one time ... an a_n probability that *Radar_Target_Identified* appears n times.

Given a specified range for scope, the typical expression:

(* <n_1-n_2> Radar_Target_Identified *)

Becomes:

(* <n_1-n_2/a_0,a_1,a_2,a_3 ... a_{n2-n1}> Radar_Target_Identified *)

There is an a_0 probability that *Radar_Target_Identified* appears n_1 times, an a_1 probability that *Radar_Target_Identified* appears (n_1+1) times... an a_{n2-n1} probability that *Radar_Target_Identified* appears n_2 times.

Eagle6 Exhaustive Scenario Generation

The following graphic represents the Eagle6 exhaustive scenario generation user interface options page. The options page has the ability to refine the application output by setting parameters for three general options:

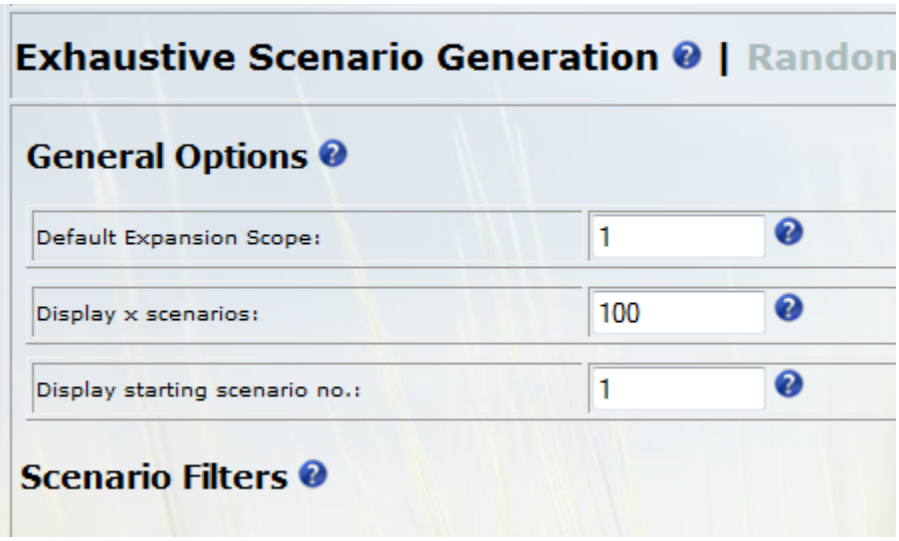


Figure 49: Exhaustive Scenario Generation Options

The following model demonstrates how to set the probability of an event. In the following test, it was determined that the event Radar_Target_Identified had two possible outcomes: Enemy_Target and Friendly_Target. To better model the operational environment, the modeler assigned the probability of an Enemy_Target being identified 60% more often than a Friendly_Target.

EXHAUSTIVE GENERATION DEMONSTRATION:

Consider the following Model:

ROOT Radar_Target_Identified: (

(Enemy_Target | Friendly_Target)

[(In_Weapon_Range Target_Lock)]

(* <1-3> Target_Ready_For_Fire *));

Set following probabilities:

- 60% probability of event Enemy_Target happening instead of event Friendly_Target
- 33.3% probability of events In_Weapon_Range and Target_Lock to appear
- 20% probability of event Target_Ready_For_Fire to appear one time
- 30% probability of event Target_Ready_For_Fire to appear two times
- 50% probability of event Target_Ready_For_Fire to appear three times

The following model represents the system modeling requirements:

ROOT Radar_Target_Identified: (

(Enemy_Target | <0.40> Friendly_Target)

[<0.33> (In_Weapon_Range Target_Lock)]

(*<1-3/0.20,0.30,0.50> Target_Ready_For_Fire *));

The following filter was applied to the Radar_Target_Identified model:

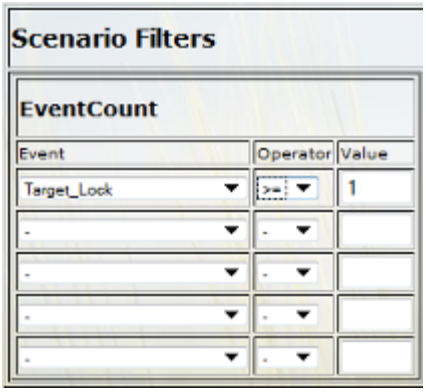
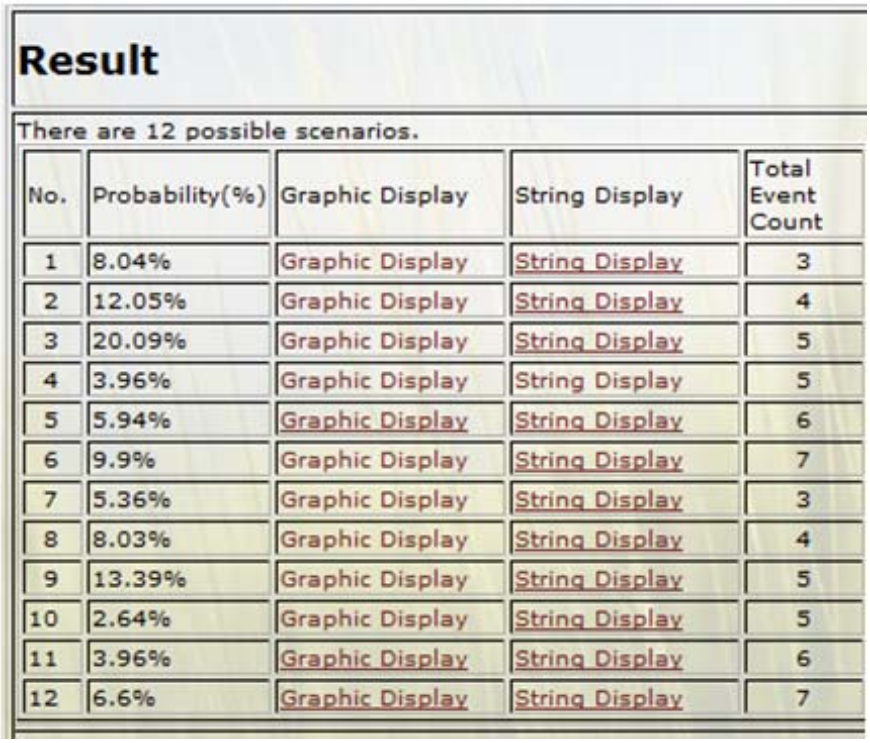


Figure 50: Radar_Target_Identified Filter

Model Results

The graphic displays the probability of all possible scenarios using the exhaustive scenario generation approach.



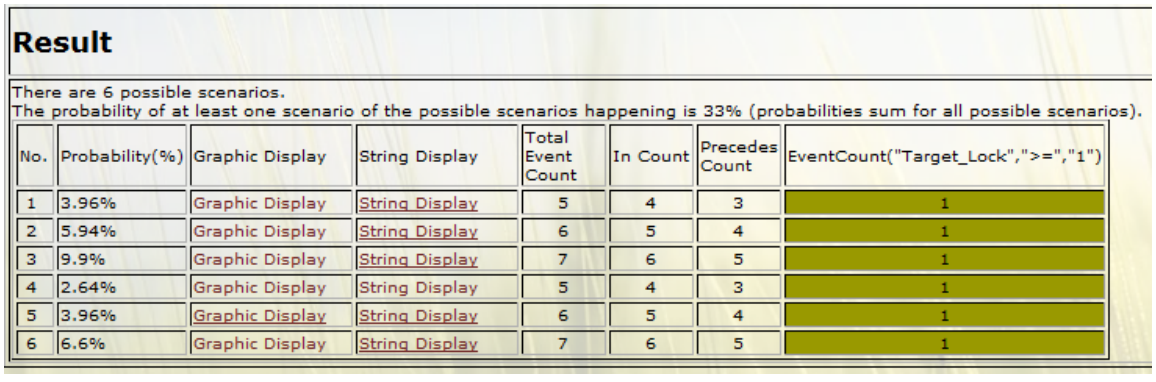
Result

There are 12 possible scenarios.

No.	Probability(%)	Graphic Display	String Display	Total Event Count
1	8.04%	Graphic Display	String Display	3
2	12.05%	Graphic Display	String Display	4
3	20.09%	Graphic Display	String Display	5
4	3.96%	Graphic Display	String Display	5
5	5.94%	Graphic Display	String Display	6
6	9.9%	Graphic Display	String Display	7
7	5.36%	Graphic Display	String Display	3
8	8.03%	Graphic Display	String Display	4
9	13.39%	Graphic Display	String Display	5
10	2.64%	Graphic Display	String Display	5
11	3.96%	Graphic Display	String Display	6
12	6.6%	Graphic Display	String Display	7

Figure 51: Model Results Showing Probability

Note: If the user selects query criteria on the options page, the result set may contain probability values for scenarios that do not total 100%. This is due to possible scenarios having been filtered from the final result set:



Result

There are 6 possible scenarios.
The probability of at least one scenario of the possible scenarios happening is 33% (probabilities sum for all possible scenarios).

No.	Probability(%)	Graphic Display	String Display	Total Event Count	In Count	Precedes Count	EventCount("Target_Lock",">=","1")
1	3.96%	Graphic Display	String Display	5	4	3	1
2	5.94%	Graphic Display	String Display	6	5	4	1
3	9.9%	Graphic Display	String Display	7	6	5	1
4	2.64%	Graphic Display	String Display	5	4	3	1
5	3.96%	Graphic Display	String Display	6	5	4	1
6	6.6%	Graphic Display	String Display	7	6	5	1

Figure 52: Model Results Showing Probability

The results of the Radar_Target_Identified event, after applying the filter, were a record set of six possible scenarios, with a probability of 33% that one of the six events will occur.

RANDOM GENERATION DEMONSTRATION

The model is an exact copy of the model used in the exhaustive scenario generation method demonstrated:

Consider the following Model:

ROOT Radar_Target_Identified: (

(Enemy_Target | Friendly_Target)

[(In_Weapon_Range Target_Lock)]

(* <1-3> Target_Ready_For_Fire *));

Set following probabilities:

- 60% probability of event Enemy_Target happening instead of event Friendly_Target
- 33.3% probability of events In_Weapon_Range and Target_Lock to appear
- 20% probability of event Target_Ready_For_Fire to appear one time
- 30% probability of event Target_Ready_For_Fire to appear two times
- 50% probability of event Target_Ready_For_Fire to appear three times

The following model represents the system modeling requirements:

ROOT Radar_Target_Identified: (

(Enemy_Target | <0.40> Friendly_Target)

[<0.33> (In_Weapon_Range Target_Lock)]

(*<1-3/0.20,0.30,0.50> Target_Ready_For_Fire *));

Demonstration:

To generate random scenario generation, the user must select the "Random Scenario Generation" link at the top of the options page:

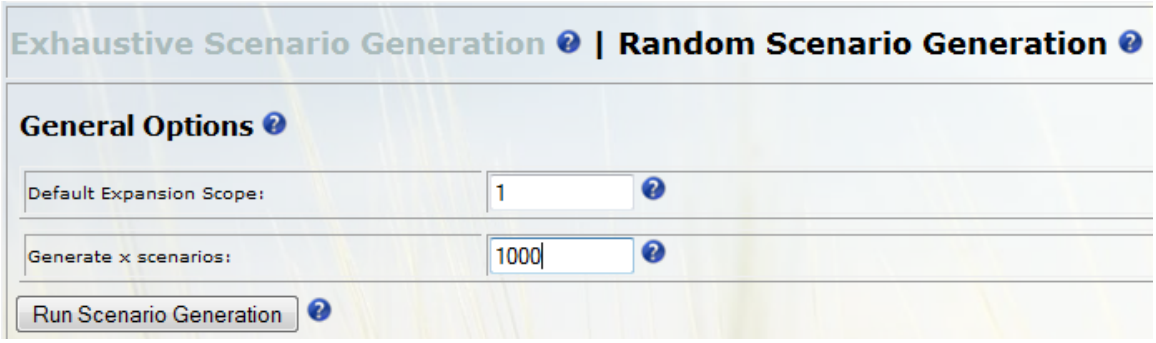
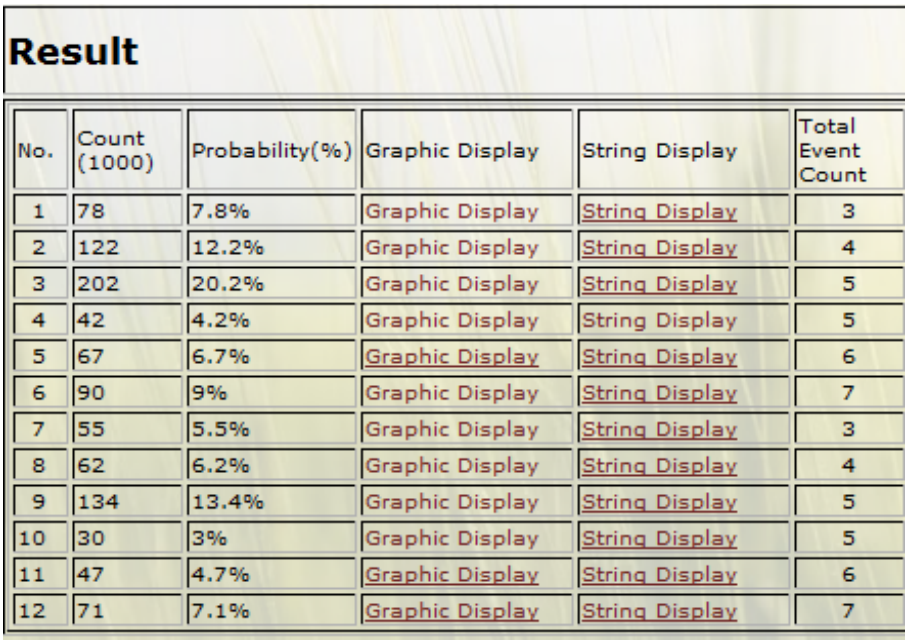


Figure 53: Random Scenario Generation Options

Model Results

In the following graphic, Eagle6 displays how many times each scenario was generated and the probability for each scenario (calculated using the total number of scenarios and the number of times each scenario appeared).



Result

No.	Count (1000)	Probability(%)	Graphic Display	String Display	Total Event Count
1	78	7.8%	Graphic Display	String Display	3
2	122	12.2%	Graphic Display	String Display	4
3	202	20.2%	Graphic Display	String Display	5
4	42	4.2%	Graphic Display	String Display	5
5	67	6.7%	Graphic Display	String Display	6
6	90	9%	Graphic Display	String Display	7
7	55	5.5%	Graphic Display	String Display	3
8	62	6.2%	Graphic Display	String Display	4
9	134	13.4%	Graphic Display	String Display	5
10	30	3%	Graphic Display	String Display	5
11	47	4.7%	Graphic Display	String Display	6
12	71	7.1%	Graphic Display	String Display	7

Figure 54: Model Results Showing Probability for 1000 Generated Scenarios

H. DEMONSTRATION SUMMARY

The demonstrations 1-5 show how the Eagle6 application may improve the current method in which the SSSTRP executes Software Safety assessments. Demonstrations six and seven demonstrate the ability for Eagle6 to test functional requirements, which is also a part of the SSSTRP process. The following Hazard State conditions were created to demonstrate the application of Eagle6 to the Software Safety domain, with specific applicability to the SSSTRP process:

- Find scenarios where the gun weapon system may require more watts than the gun weapon system can produce.
- Find scenarios where the gun weapon system may require more network bandwidth than the gun weapon system network can provide.
- Find scenarios where the gun weapon system may experience the failure of a Gun Control Center Open Fire Command.

- Find a set of scenarios that contain at least one GCC_openFire event, and also have at least one slice of events that have the Attribute Req_Num_Man_Approv_For_Cmd with a sum that is >= 3.
- Find scenarios where Gun Console Computer tries to execute an Open Fire Command and it ends with a system timeout.

The demonstrations 1-5 show how the system and environment can be modeled with specific focus on the ability to model system behavior. This capability is especially helpful in the SSSTRP software safety domain as the need exists to not only check for potential software hazard states, but also create domain models that enable the testing of potential software with realistic environmental events, and the probabilities associated with those events. This approach is much more refined and allows for domain models to better reflect the operational behavior in which the systems function. With a methodology for modeling system behavior, and an ability to generate estimates for both functional and nonfunctional requirements, the next step is to identify how the proposed methodology can be integrated into the SSSTRP process.

I. PROTOTYPE SSSTRP EVALUATION METHODOLOGY

The prototype SSSTRP evaluation methodology is based on the research of the current SSSTRP process (Chapter I), the problems associated with the SSSTRP evaluation process (Chapter II), and Eagle6 capabilities that are demonstrated in Chapter III). The prototype has not been tested, but is included in this research based on the relevance to the domain. The purpose of the prototype SSSTRP evaluation methodology is to recommend changes to the current SSSTRP process that includes the integration of our modeling methodology, as well as a more definitive evaluation process.

Our research demonstrates that Eagle6 was able to provide the ability to model potential systems and how they interact with their environment. The Eagle6 solution requires an integration plan that introduces the methodology into

the SSSTRP process. The prototype SSSTRP evaluation methodology is recommended for integrating a standardized methodology for automating system testing.

The solution to model a gun weapon system and to have the ability to test for software safety assertions was addressed in Chapter III. The Prototype SSSTRP Evaluation Methodology is designed to obtain the functional and nonfunctional requirements of a system before the acquisition community has released the RFP. This change in process allows for the development of a TDP questionnaire that is designed to elicit responses that can be entered into a model and evaluated. The SSSTRP Evaluation Methodology has five major components:

Step 1: Develop Domain Model

Purpose: To develop a domain model of the current system.

Expected Benefits: The MP model enables the SSSTRP to model the current system state in order to evaluate proposed changes to the system.

Artifacts:

- MP Model
- List of Hazard States
- Assertion Library
- Functional Requirements
- Nonfunctional Requirements

Step 2: Develop Vendor Questionnaire

Purpose: The Vendor Questionnaire is designed to elicit responses to questions about system behavior. Formatting the Vendor Questionnaire in such a way that requires the vendor to respond with measurable answers enables the evaluation of the TDP to be automated.

Expected Benefits: MP does not require specific knowledge of systems in order to model a system. This abstract approach to SoS modeling supports a vendor questionnaire structure that is designed to elicit answers that reflect the compatibility of the proposed system relative to the current operational environment.

Artifacts:

Vendor Questionnaire

Step 3: Organize Vendor TDP Response

Purpose: Organizing the Vendor TDP response requires the TDP answers to be formatted in a standardized way that can be input into the domain model.

Expected Benefits: Standardizing the TDP response into data that can be automatically read into a domain model reduces risk of human error and standardizes the results that are output by the model.

Artifacts:

TDP Model Input Files

Step 4: Formally Evaluate Software

Purpose: To execute test plan and generate results in graphical and textual formats. The results of the tests are formatted and given to the SSSTRP for evaluation.

Expected Benefits: Executing standardized test plans enables the SSSTRP to evaluate the results by generating the following reports:

- Comparison Analysis - Compares the system side-by-side to create an evaluation of the proposed systems in a consolidated format.
- Assertion Checking Reports - Reports that identify system scenarios where assertion violations were found.

- Functional and Nonfunctional System Performance Reports - Reports that show how the system may perform when integrated as part of an overall SoS.

Artifacts:

- Comparison Analysis
- Assertion-Checking Reports
- Functional System Performance Reports
- Nonfunctional System Performance Reports

Step 5: Conduct SSSTRP Review

The SSSTRP reviews the results of the tests and requests additional tests if needed. SSSTRP findings, reports, and recommendations are then forwarded to the WSESRB for final determination.

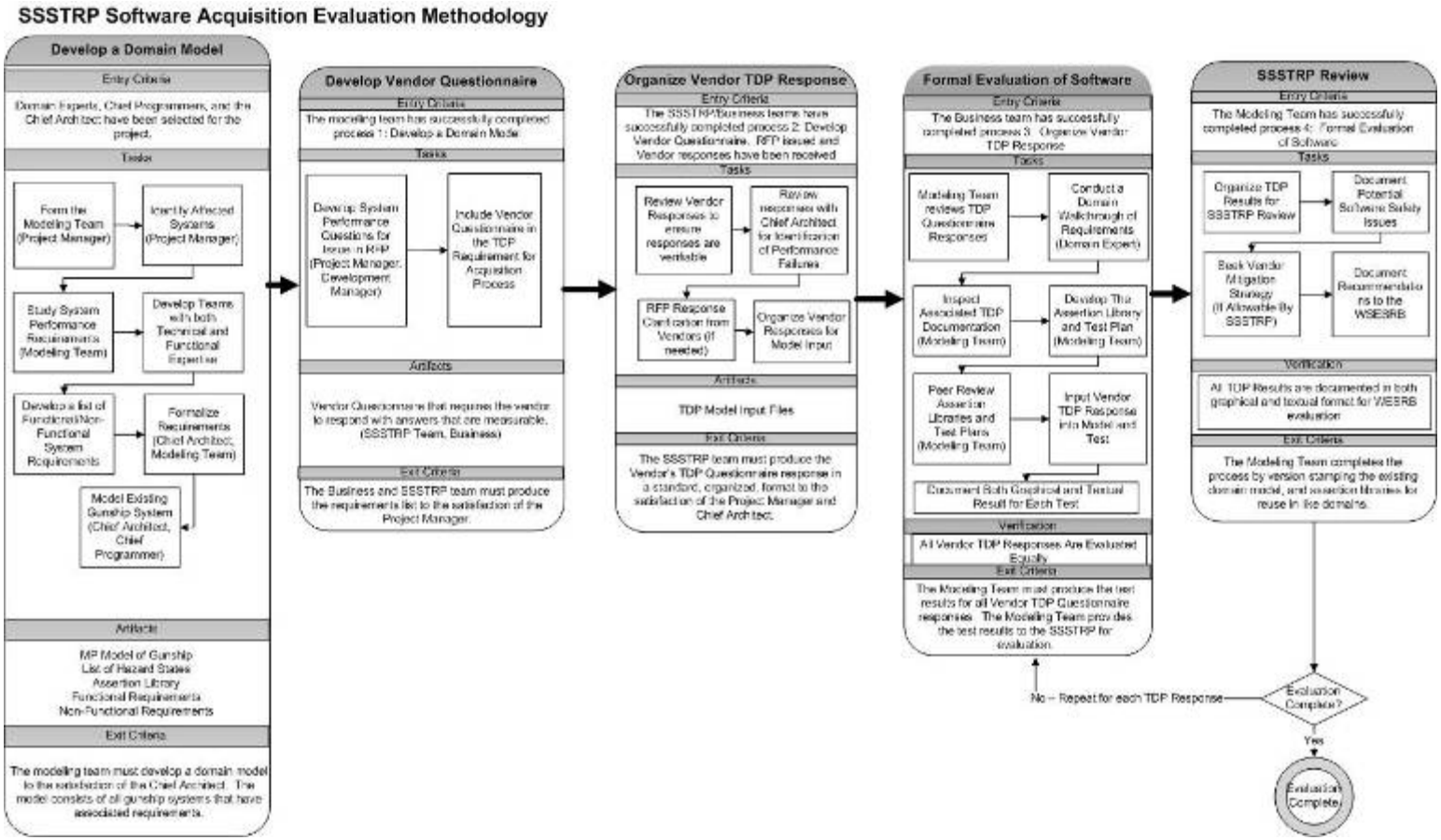


Figure 55: Proposed SSSTRP Evaluation Methodology

J. SUMMARY

Providing the SSSTRP community with high-level models that may satisfy a portion of the software safety assessment process improves the current inspection-based evaluation methodology. Without a high-level modeling process, the alternative is to implement the system and to perform testing. Manual testing is a very expensive and timely alternative, which may be partially satisfied using the prototype methodology and tools that are covered in this chapter.

The Prototype SSSTRP Evaluation Methodology is designed to obtain the functional and nonfunctional requirements of a system before the acquisition community has released the RFP. This change in process allows for the development of a TDP questionnaire that is designed to elicit responses that can be entered into a model and evaluated. The revised SSSTRP process includes artifacts that supports a more structured evaluation process.

K. LIMITATIONS OF THE PROTOTYPE SSSTRP PROCESS

The proposed revised SSSTRP process introduces the results of this research into the current SSSTRP process while adding artifacts within each evaluation stage. The proposed changes to the SSSTRP process have not been tested within the SSSTRP process; therefore the validity of the proposed process is unknown.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. EAGLE6–PROTOTYPE SOFTWARE ARCHITECTURE MODELING SOFTWARE

The need for graphical representation of system models required automated tools to compile and display traces of model execution in textual and graphical formats. The demonstration of Eagle6 for naval gun weapon system software was achieved by developing custom software with the following components.

- Custom software (compiler/lexer/parser) to process MP models.
- Custom software that displays the MP model in textual and graphical formats.
- Dynamic Query interface that enables the user to return a set of scenarios based on an iterative or random scenario generation approach. These approaches are described later in this section.

The MP model used to demonstrate the modeling software can be found in Appendix A.

The Eagle6 application was designed and built with programming help from Alex Gociu.

A. EAGLE6 PROTOTYPE SOFTWARE ARCHITECTURE

Eagle6 modeling software consists of the following functionality:

- Parse and validate modeling language
- Generate all possible scenarios within scope
- Build dynamic queries
- Display scenarios graphically
- Export scenarios to text
- Provide detailed scenario runtime output

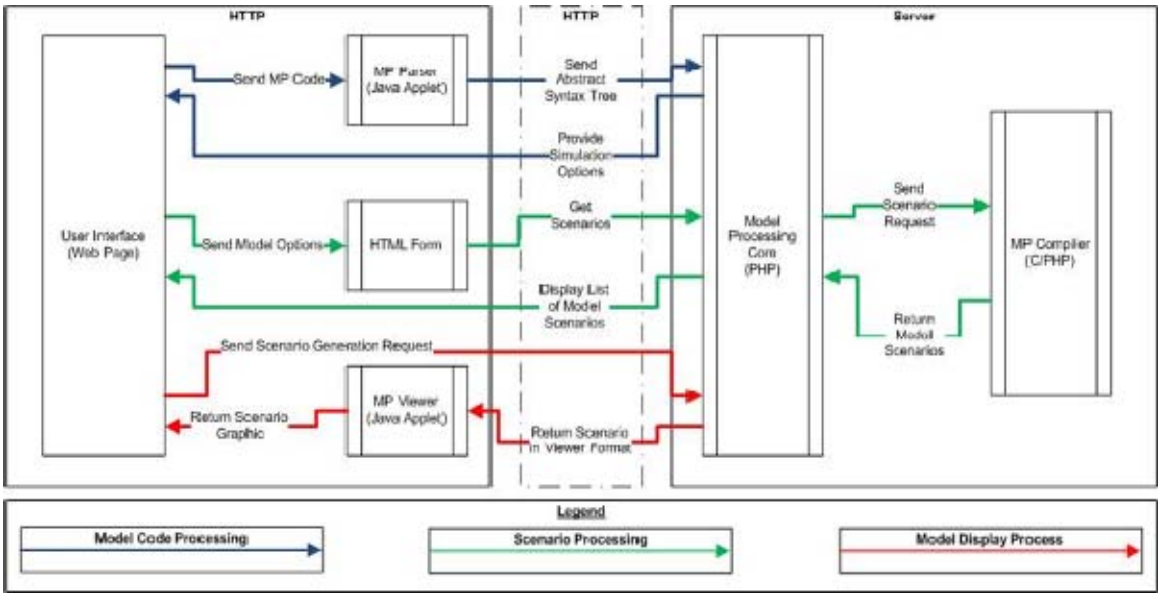


Figure 56: Eagle6 Prototype Software Architecture

The Eagle6 Prototype Software diagram model represents the Eagle6 system architecture.

B. EAGLE6 PROTOTYPE SOFTWARE DIAGRAM

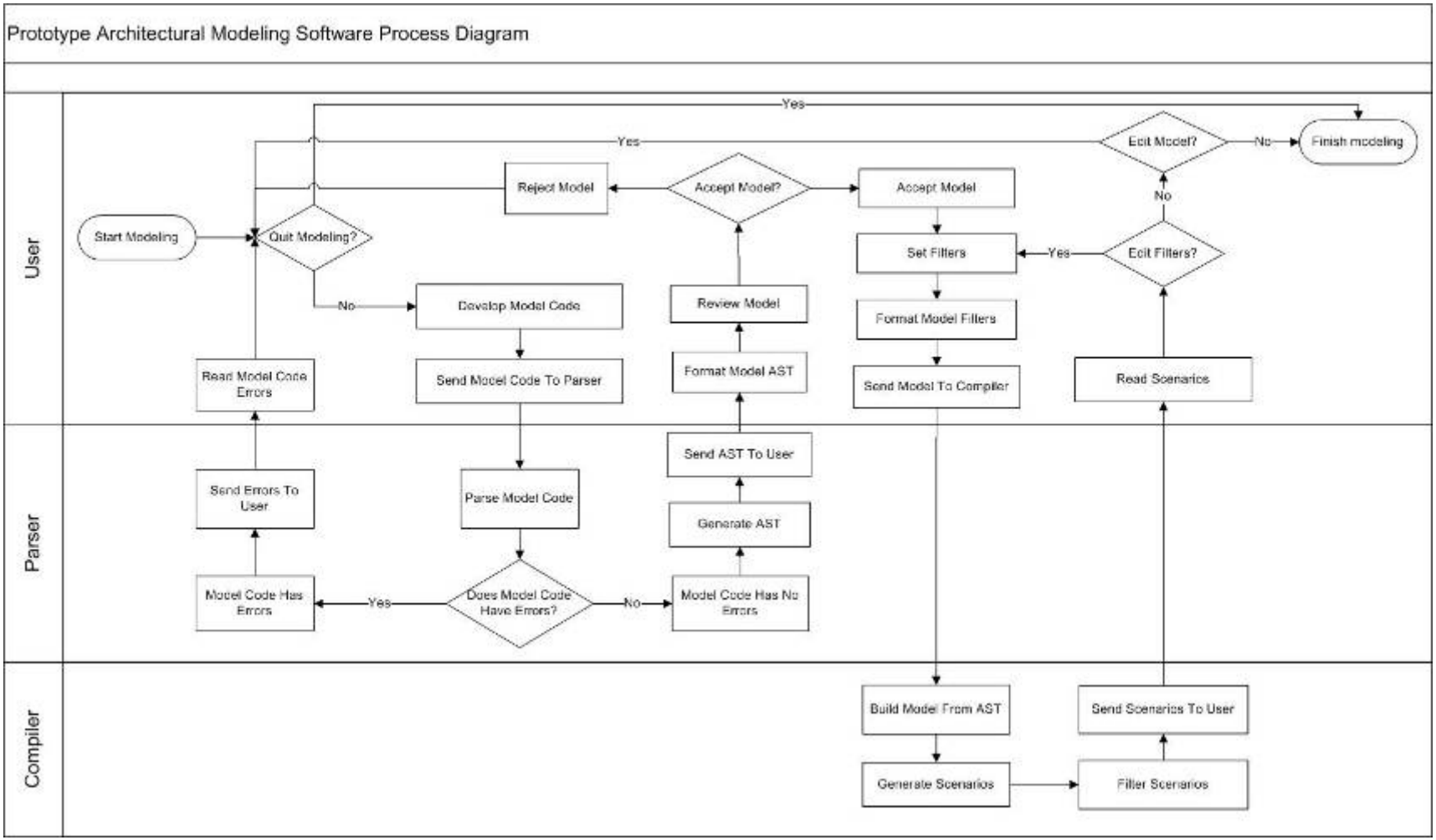


Figure 57: Eagle6 User Experience Model

C. MP MODEL OF INTERACTION BETWEEN EAGLE6 AND USER

The following MP model demonstrates the ability to model system design as demonstrated in the "Eagle6 User Experience Model."

```
ROOT User_activity: (
        StartModeling
        (*
                DevelopModelCode
                SendModelCodeToParser
                (
                        (
                                SendErrorsToUser
                                ReadModelCodeErrors
                        )
                        | (
                                SendAstToUser
                                FormatModelAst
                                ReviewModel
                                (
                                        RejectModel
                                        | (
                                                AcceptModel
                                                (*
                                                        SetFilters
                                                        FormatModelFilters
                                                        SendModelToCompiler
                                                        SendScenariosToUser
                                                        ReadScenarios
                                                *)
                                        )
                                )
                        )
                )
        *)
        FinishModeling
);

ROOT Parser_activity: (*
        SendModelCodeToParser
        ParseModelCode
        (
                (
                        ModelCodeHasErrors
                        SendErrorsToUser
                )
                | (
                        ModelCodeHasNoErrors
                        GenerateAst
                        SendAstToUser
                )
        )
*);
```

```
ROOT Compiler_activity: (*
        SendModelToCompiler
        BuildModelFromAst
        GenerateScenarios
        FilterScenarios
        SendScenariosToUser
*);

Parser_activity, User_activity SHARE ALL SendModelCodeToParser, SendErrorsToUser,
SendAstToUser;
Compiler_activity, User_activity SHARE ALL SendModelToCompiler, SendScenariosToUser;
```

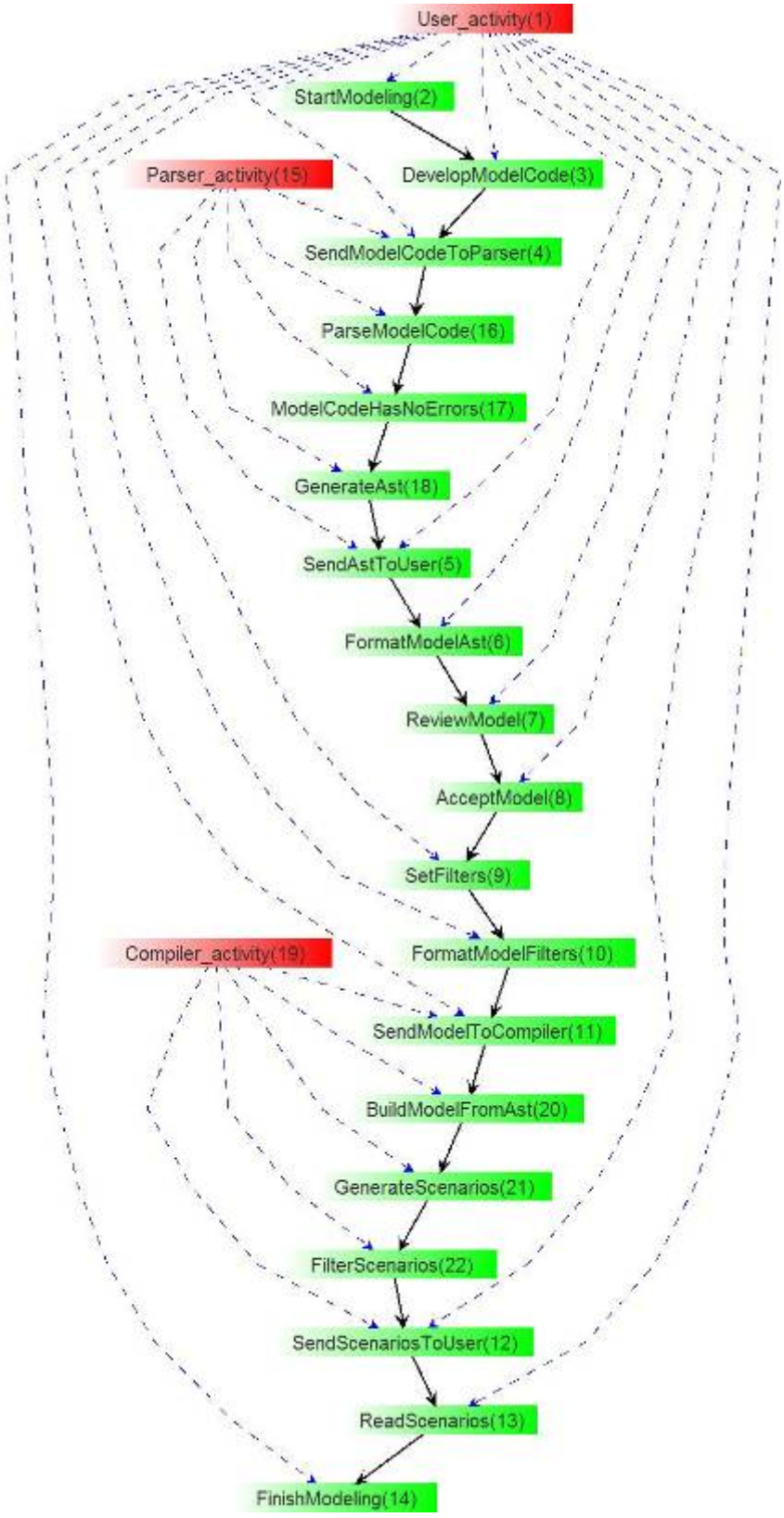


Figure 58: Eagle6 MP Architecture Scenario

The Eagle6 MP Architecture Scenario represents the Eagle6 system architecture. MP has the capability to quickly modify existing schemas to be used in future system architecture verification.

D. PROTOTYPE COMPILER ARCHITECTURE

The system has four major components:

- User Interface (located on web server)
- Model Compiler (located on web server)
- Model Parser (java applet)
- Model Viewer (java applet)

1. Eagle6 Compiler Design

The Prototype Model Compiler is built in C++ and is located on a web server. The compiler has the following functionality:

- Interact with the User
    - Provide HTML graphical interface for the user
    - Provide model parser and viewer Java applets
    - Get abstract syntax tree from model
    - Provide and get simulation options
    - Provide scenarios list
    - Export scenarios to different formats
- Interact with the Eagle6 Model Compiler
    - Send model program and simulation options
    - Create a set of all possible scenarios within scope

2. Eagle6 Lexer and Parser

The Eagle6 Model Parser provides the functions of a lexer and parser for the MP language (Auguston, Software architecture built from behavior models, 2009). The Eagle6 Model Parser receives user input representing an MP model, validates the code, returns syntax error information (if applicable), and builds the abstract syntax tree. The Eagle6 Model Parser is built in Java and is a Java applet.

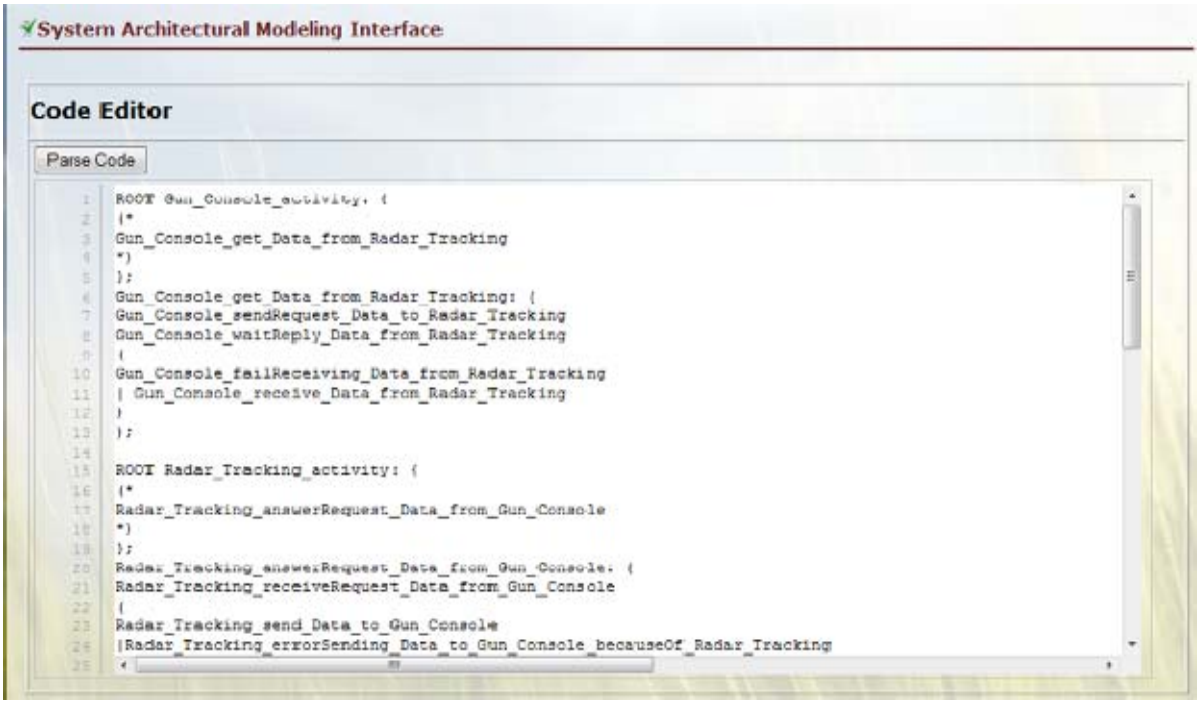


Figure 59: Prototype MP Editor

E. EAGLE6 PROTOTYPE PARSER AND HELPER

The “Parse Code” button executes the MP Model Code Parser that checks for syntax errors in the model. If the model structure is incorrect, an error message is displayed that identifies the specifics of the error. The following is an example of an error message:

Figure 60: Eagle6 Prototype Parser Error Handling

If the code is parsed successfully, the user is presented with a page that allows the user to further define the criteria for generating scenarios.

F. EAGLE6 PROTOTYPE VIEWER FOR GRAPHICAL AND TEXTUAL DISPLAY OF SCENARIO

The Eagle6 Prototype Viewer is built in Java and is a Java applet with JGraph being the application used for graphical representation of the model. The scenario generation options page allows for models to be generated and tested with a full degree of fidelity using the detail display options.

1. Eagle6 Prototype Viewer General Options

The scenario generation options page is used to set the parameters of your test. This page has the following characteristics: (1) The general options of your test can be set to include the expansion scope and the level of details returned from the compiler; and (2) scenario filter conditions represent dynamic queries that are used to set filtering parameters, so the results returned by the compiler represent the user's target test data.

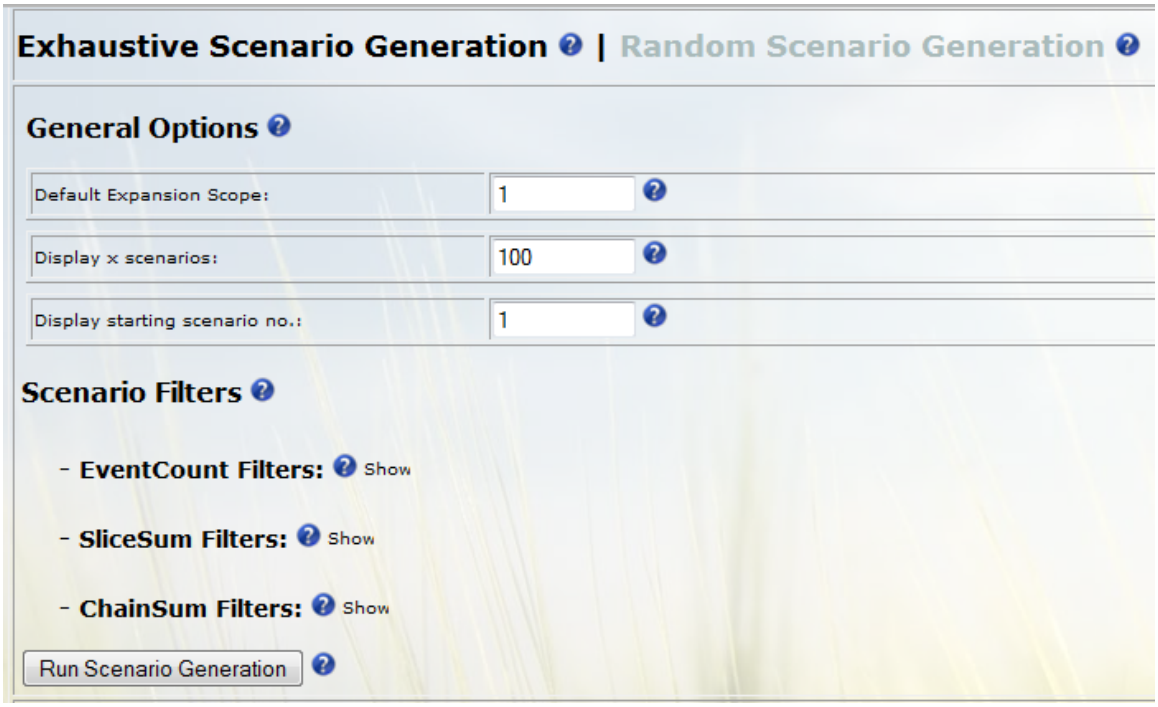


Figure 61: Eagle6 Prototype Viewer Scenario Generator

Default Expansion Scope – The purpose of the default expansion scope is to limit the size of the "*" rule in order to better define the scenario's parameters. For example, if the test scenario requires the gun weapon system to fire three rounds, the scenario's scope is set to "3," thereby removing the infinite ("*") default parameter. In the absence of an expansion scope, setting this value will result in a finite number of scenarios.

Display x scenarios – Defines the total number of scenarios to be displayed.

Display starting scenario no. – Displays scenarios starting at a specific number. The option of generating a list starting at a specific number allows for a streamlined architecture verification process.

2. Eagle6 Prototype Viewer Scenario Generation Filter

In order to refine the models returned from the compiler, it is necessary to refine the data inputs that are used by the compiler in order to filter scenario results. Eagle6 uses a dynamic query builder to satisfy this requirement, as shown in the event count conditions function:

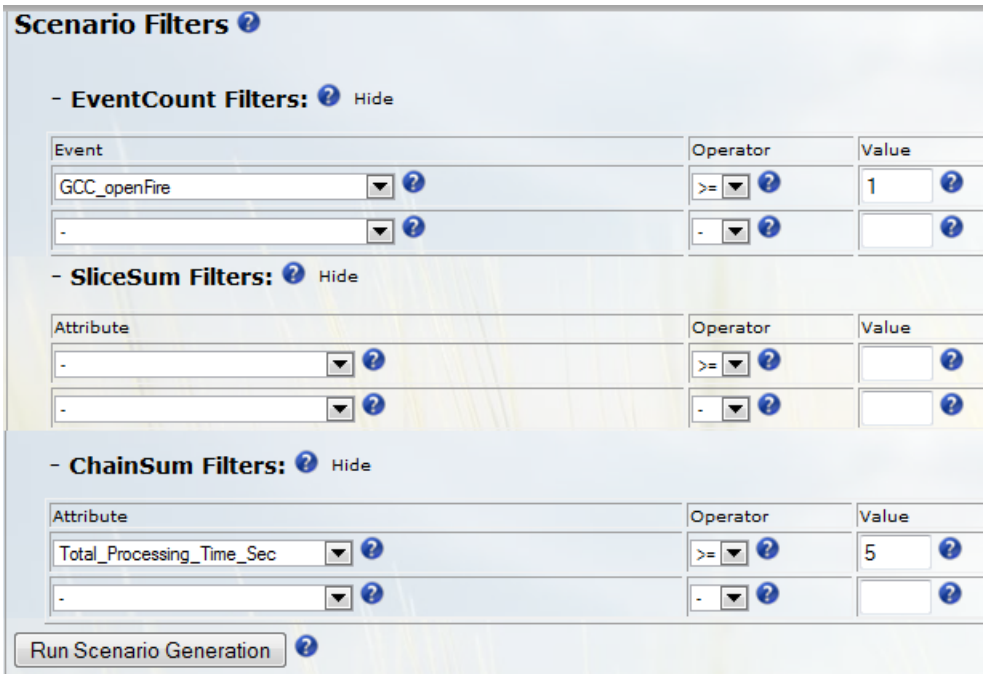


Figure 62: Eagle6 Prototype Viewer Scenario Generator Filter

EventCount (event count) – Enables the user to refine the results returned from the compiler by limiting the results to scenarios that have a specific event, and event count condition.

- Event – Event is used to select a specific event that is supplied to the dynamic query builder.
- Operator – Sets the evaluation parameters for the query builder. Values are: "<", "<=", ", "=", ">", ">="
- Value – Value sets the specific event count parameters used by the query builder.

SliceSum (maximum slice sum) – SliceSum is used to find scenarios that contain events that run in parallel and have attribute values that, when summed, meet the query builder criteria.

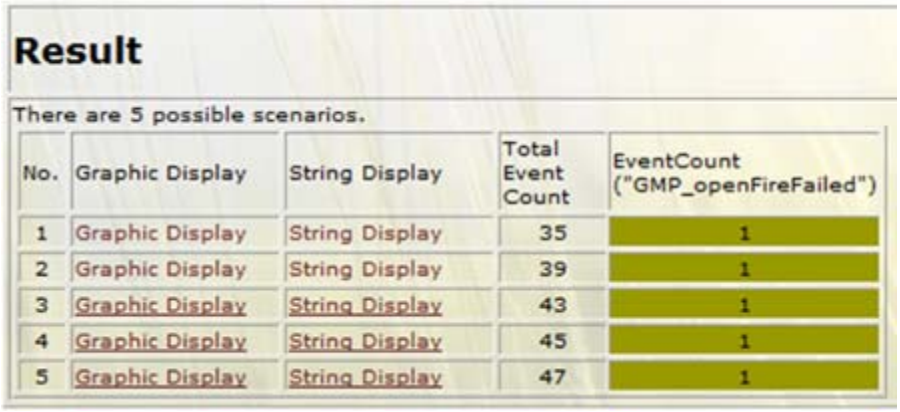
- Attribute – User-defined event attribute that is identified in the system model.
- Operator – Sets the evaluation parameters for the query builder. Values are: "<", "<=", "", "=", ">", ">="
- Value – Value sets the specific event count parameters used by the query builder.

ChainSum – ChainSum is used to find scenarios that contain events that run in sequence and have attribute values that, when summed, meet the query builder criteria.

- Attribute – User-defined event attribute that is identified in the system model.
- Operator – Sets the evaluation parameters for the query builder. Values are: "<", "<=", "", "=", ">", ">="
- Value – Value sets the specific event count parameters used by the query builder.

Run Simulation

The program will calculate all possible scenarios within scope, and display a list of scenarios for the user to view:



Result

There are 5 possible scenarios.

No.	Graphic Display	String Display	Total Event Count	EventCount ("GMP_openFireFailed")
1	Graphic Display	String Display	35	1
2	Graphic Display	String Display	39	1
3	Graphic Display	String Display	43	1
4	Graphic Display	String Display	45	1
5	Graphic Display	String Display	47	1

Figure 63: Eagle6 Prototype View Scenario Generator Result

The results of the test were five possible scenarios. Using the “Show Details” function, the number of events and number of relationships from each scenario are also displayed. The scenario test can be viewed by the string or graphical display options.

The Eagle6 Prototype Viewer is used for displaying scenarios in graphical representation. Events are the vertices and the IN/PRECEDES relations between them, noted by arrows.

Using the graphical interface, the user is able to filter event views. For example, the following graphic represents the scenario with associated events and connectors:

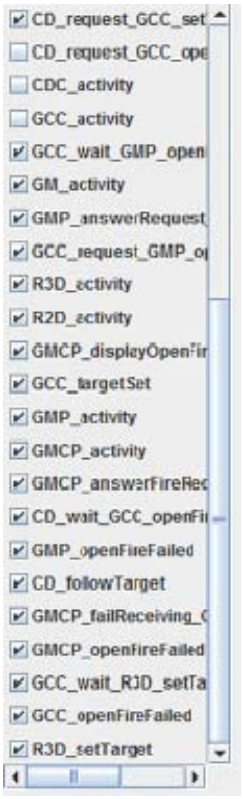


Figure 64: Eagle6 Prototype Viewer Filter Functionality

The result of the filter interface is the hiding of all unselected events, which allows for a more readable graphic display of the scenario.

G. LIMITATION OF EAGLE6 TOOL

The Eagle6 tool is considered a prototype and has not been tested using multiple case studies and test tools. The tool has the following limitations:

- Tool Verification – The tool does not have the capability to formally verify the MP model represents the current software system architecture. Inspection techniques that compare system behavior with scenarios that are generated from custom queries are the current method for verifying models. However, it seems logical that log tools that run in the system architecture can be extracted and compared to the MP model. This is an area that has been

identified as future work, and is on the development plan for an Eagle6 future version.

- Determining the “Right” Scope – Determining the proper scope that meets the criteria for returning the maximum amount of assertion violations is largely dependent upon the complexity and purpose of the model. We cannot guarantee that the Small Scope Hypothesis will detect a majority of errors. However, given the current state of the SSSTRP evaluation process that uses inspection techniques with arbitrary test cases, the use of a tool that can test hundreds of scenarios is an improvement of the current software safety evaluation process. A proper scope is dependent upon the situation and the relative risk. Future work is required to estimate the proper scope.
- Complex Scope Computing – Enterprise models that have exponential possibilities of scenarios result in a risk of the computing power not being able to produce an acceptable number of scenarios with the Small Scope.
- Abstraction Layer Definitions – The tool does not have the ability to standardize the layers of abstraction. However, the tool does give the user the capability to customize/filter the visual representation of the scenario within the visualization tool. The current level of abstraction is defined by the User’s decision for what events they want to see.
- Architecture Modeling Versus Software/System Testing – The tool is meant to be used for software/system architecture and testing and is not meant to be used for system testing.
- Abstraction Risks for Loss of Critical System Behavior – The concept of abstraction means that assumptions have to be made about certain details of the software system. Modeling at the

abstraction layer contains a risk of developing a model that does not include critical details of the system. The MP construct allows for certain attributes to be modeled, but not all aspects for system behavior can be modeled at the abstract level.

- Statistical Evaluation in Software Safety – The process of generating random scenarios and calculating probabilities for events implies uncertainty, which may be unacceptable for some software safety assessments.
- IF and WHERE Constructs are not available – The current toolset does not allow for conditional evaluation during the scenario generation. This concept is in design and is expected to be in a future release. The WHEN handler is discussed in the ICCRTS 2010 paper by Auguston/Whitcomb entitled "System Architecture Specification Based on Behavior Models," in Proceedings of the 15th ICCRTS Conference (International Command and Control Research and Technology Symposium), Santa Monica, CA, June 22-24, 2010.
- Dynamic Attributes – Attributes that require a dynamic state cannot be modeled within the current tool and is reserved for future work.
- Finite vs. Infinite System Modeling – The current tool does not support modeling systems that do not have a finite execution.
- Limitations of the Prototype SSSTRP Process – The proposed revised SSSTRP process introduces the results of this research into the current SSSTRP process while adding artifacts within each evaluation stage. The proposed changes to the SSSTRP process have not been tested within the SSSTRP process; therefore the validity of the proposed process is unknown.
- Risks of Jackson's Small Scope Hypothesis – Determining appropriate scope levels that satisfy architecture verification is

dependent upon the situation and complexity of the model. Jackson's Small Scope Hypothesis is incapable of being verified as the definition of "Small Scope" is not verifiable. Future work is required to determine appropriate scope requirements for software safety assessments.

- Risk Assessment Capability – The Eagle6 tool is not designed to evaluate risk, and to combine the probability of events and risk to create a Risk Assessment. This issue is identified as future work.

THIS PAGE INTENTIONALLY LEFT BLANK

V. RESEARCH CONCLUSION AND CONTRIBUTIONS

The objective of the research was to identify the problems associated with the high number of SSSTRP failures. The research included a review of three years of unclassified SSSTRP reports, and an analysis of the failures (Chapter II). A prototype modeling methodology, and the ability to apply the modeling methodology to the software safety domain, was demonstrated in Chapter III.

The initial Eagle6 prototype modeling methodology framework was tested using a case study of a naval gun weapon system, found in Appendix A. The Eagle6 tool can generate executable code that can be evaluated using macro queries, which improves the current SSSTRP evaluation methodology found in Chapter II. An ability to transfer simple, abstract modeling techniques into formal methods that are able to be tested was created. The following is a summary of my research contribution relative to the improvements of the current SSSTRP process of evaluating potential gun weapon system architectural changes:

Methodology and Tools to Support Software System Safety Analysis for the SSSTRP Evaluation Process - The Eagle6 tool gives the SSSTRP modeler the ability to model the interaction between the system and its environment, as demonstrated in Chapter III. The ability to model environmental effects on software/systems enables the SSSTRP member to evaluate potential gun weapon system changes with higher fidelity compared to the current evaluation process.

Higher Fidelity of SSSTRP Evaluation via Assertion Checking – The current SSSTRP evaluation methodology is random testing using inspection techniques, as identified Chapter II. The Eagle6 tool enables the SSSTRP to create assertions about specific components and behaviors of a system, and gives them the tools to verify the assertions via formal queries.

Two Modes of Scenario Generation – Our modeling tool enables the SSSTRP to perform an exhaustive search for model verification within scope,

and Random scenario generation for statistical estimates of nonfunctional requirements, such as performance.

Extension of Monterey Phoenix Modeling Methodology - Our research extended the MP framework by using predefines macro queries (concept of “Chain” and predefine aggregate operations over events).

A. FUTURE RESEARCH OPPORTUNITIES

Research opportunities have arisen within this project but have not been fully explored. These issues are related to the SSSTRP process. Suggested areas for future research include:

- Business Process Reengineering–Eagle6 produces abstract views of systems; prototyping proposed BPR solutions could be researched.
- Oracle/Black Box Testing--Eagle6 does not require the input of system specifics. Future research opportunities exist to test system architectures that have black box components.
- Refinement of the Model Compiler – The current compiler has hardware limitations that may be improved by improving the hardware processing capability, and the compiler software design.
- Graphical User Interface for Model Abstraction – A graphical design tool that automatically generates model code could be developed. A visual interface that allows dynamic addition of systems (and subsystems), and connections between them, would improve the speed of development.
- Development of a methodology that encompasses the evaluation of Dynamic Attribute Values - Evaluation of dynamic attribute values is necessary, since the events of a system often cause chain reactions that could change an attribute's value.

- Development of the SSSTRP Evaluation Process – This dissertation describes a suggested methodology for implementing our methodology into the SSSTRP Software evaluation process. Further research that is focused on the process and implementation of a formal method for evaluating software is needed.
- Risk Assessment – Capturing the measured consequence of an event, and combining it with the probability of the event, should lead to some form of Risk Assessment.
- Model Verification – An ability to verify the accuracy of an MP model is certainly an area for future research.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Abowd, G., Alen, R., & Garlan, D. (1995). Formalizing style to understand descriptions of software architecture. *ACM Transactions on Software Engineering and Methodology*, 319–364.

Allen, R. (1997). *A formal approach to software architecture.* Pittsburg: Ph.D. Thesis, Carnegie Mellon University, CMU Technical Report.

Allen, R., & Garlan, D. (1997). A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 213–249.

Archer, M., Lim, H., Mitra, S., Lynch, N., & Umeno, S. (2008). Specifying and proving properties of timed I/O automata using tempo. *Design Automation for Embedded Systems*, 1–2.

Auguston, M. (1995). Program behavior model based on event grammar and its application for debugging automation. In *2nd International Workshop on Automated and Algorithmic Debugging*, (pp. 277–291).

Auguston, M. (2009). Software architecture built from behavior models. *ACM SIGSOFT Software Engineering Notes*, 34–39.

Auguston, M. (2009b). Monterey phoenix, or how to make software executable. *OOPSLA 2009*, (pp. 1031–1038).

Auguston, M., & Whitcomb, C. (2010). System architecture specification based on behavior models. In *International Command and Control Research and Technology Symposium*, (pp. 1–20).

Auguston, M., Michael, B., & Shing, M.-T. (2006). Environment behavior models for automation of testing and assessment of system safety. *Information and Software Technology, Elsevier*, 971–980.

Azani, C. (2001). The test and evaluation challenges of following an open system. *ITEA Journal, 22* (3), 1-15.

Banatre, J.-P., & Metayer, D. L. (Jan). Programming by multiset transformation. *Communnications of the ACM, 36*(1), 98–111.

Barrett, P. (1993). Delta-4: An open architecture for dependable systems. *IEEE Colloquium on Safety Critical Distributed Systems* (pp. 2.1-2.7). London: IEEE.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice,* (2nd ed.). Boston: Addison-Wesley.

Bell, R. (2006). Introduction to IEC 61508. *10th Australian Workshop on Safety Critical Systems and Software* (pp. 3–12). Sydney: Australian Computer Society Inc.

Bertolino, A., & Mirandola, R. (2004). Software performance engineering and component-based systems. *4th International Workshop on Software and Performance* (pp. 238–242). Redwood Shores: ACM.

Bhansali, P. (2005). Universal software safety standard. *ACM SIGSOFT Software Engineering Notes*, 1–4.

Booch, G., Jacobson, I., & Rumbaugh, J. (2000, June 3). *OMG unified modeling language specification.* Retrieved April 4, 2008, from OMG: UML Specification: http://www.omg.org/docs/formal/00-03-01.pdf

Clements, P., & Shaw, M. (2009). The golden age of software architecture revisited. *IEE Software*, 70–72.

Coronato, A., d'Acierno, A., & De Pietro, G. (2005). Automatic implementation of constraints in component based applications. *Information and Software Technology, 47*(7), 497–509.

Department of the Navy. (2005, March 8). OA Assessment Model. Washington, DC.

Department of the Navy. (2008, June 4). *Naval Open Architecture Contract Guidebook.* Retrieved August 31, 2009, from Naval Open Architecture: https://acc.dau.mil/CommunityBrowser.aspx?id=18016&lang=en–US

England, P., Lampson, B., Manferdelli, J., Peinado, M., & Willman, B. (2003). A trusted open platform. *Computer, 36*(7), 55–63.

Fabresse, L., Dony, C., & Huchard, M. (2008). Foundations of a simple and unified component–oriented language. *Computer Languages, Systems and Strutures, 34*(2–3), 130–149.

Graph Transformation. (1997). *Handbook of graph grammars and computing.* (G. Rosenberg, Ed.) River Edge: World Scientific Publishing Company.

Hoare, C. A. (1985). *Communicating sequential processes.* New York: Prentice-Hall.

Inverardi, P., & Wolf, A. L. (1995). Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE Transactions on Software Engineering, 21*(4), 373–386.

Jackson, D. (2006). *Software abstractions: logic, language, and analysis.* Cambridge: MIT Press.

Jackson, D. (2009). A direct path to dependable software: who could fault an approach that offers greater credibility at reduced cost? *Communications of the ACM*, 78–88.

Jackson, D. (2009b). *MIT: Alloy analyzer 4.1.10*. Retrieved May 5, 2009, from Alloy Analyzer 4.1.10: http://alloy.mit.edu/community/software

Jackson, D., & Damon, C. A. (1996). Elements of style: Analyzing a software design feature with a counter example detector. *IEEE Transactions on Software Engineering*, 22.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 558–565.

Leveson, N. (1995). *Safeware: System safety and computers.* New York: Addison–Wesley.

Medikonda, B., & Panchumarthy, S. (2009). A framework for sofware safety in safety–critical systems. *SIGSOFT Software Engineering Notes, 34*(2), 1–9.

Merola, L. (2006). The COTS software obsolescence threat. *5th International Conference on Commercial–off–the–Shelf (COTS)–Based Software Systems* (pp. 127–133). New York: IEEE.

Mohamed, A., Ruhe, G., & Eberlein, A. (2007). Decision support for handeling mismatches between COTS products and system requirements. *Sixth International IEEE Conference on Commercial–off–the–Shelf (COTS)–based Software Systems* (pp. 63–72). New York: IEEE.

National Aeronautics and Space Administration. (2004). *NASA Software Safety Guidebook.* Washington, DC: NASA.

National Aeronautics and Space Administration. (2004, July 8). *NASA–STD–8719.13B.* Retrieved July 9, 2009, from Software Safety Standard: http://www.hq.nasa.gov/office/codeq/doctree/871913B.pdf

Naval Sea Systems Command. (97). *Navy Weapon System Safety Program.* Washington: NAVSEAINST 8020.6D.

Perry, D. E., & Wolf, A. L. (1992). Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 40–52.

Rajsuman, R., & Noriyuki, M. (2004). Open architecture test system: System architecture and design. *2004 International Test Conference* (pp. 403–412). New York: IEEE.

Rehman, M., Yang, X., Dong, J., & Ghafoor, M. (2005). Prioritized selecting COTS vendor in COTS–based software development process. *Canadian Conference on Electrical and Computer Engineering (CCECE)* (pp. 1839–1845). Saskatoon: IEEE.

Reussner, R., Schmidt, H., & Poernomo, I. (2003). Reliability prediction for component–based software architectures. *The Journal of Systems and Software*, 241–252.

Rivera, J. (2010). Applying architecture modeling methodology to the naval gunship software safety domain. *ACM/IEEE 13th International Conference on Model Driven Engineering Language and Systems* (pp. 43–48). Oslo, Norway: IEEE.

Rivera, J., & Luqi. (2010). *Requirements framework for the software system safety technical review panel technical review package.* Monterey: Naval Postgraduate School.

Spivey, J. M. (1992). *The z notation: a reference manual.* New York: Prentice Hall International Series in Computer Science.

Tamura, Y., Yamada, S., & Kimura, M. (2006). Sofware reliability modeling in distributed development environment. *Journal of Quality in Maintenance Engineering, 12* (4), 425–432.

U.S. Navy. (2008, January 3). *USS Forrestal (CV–59)*. Retrieved February 19, 2009, from DEPARTMENT OF THE NAVY–NAVAL HISTORICAL CENTER: http://www.history.navy.mil/danfs/f3/forrestal.htm

Wulf, V., Pipek, V., & Won, M. (2008). Component–based tailorability: Enabling gighly flexible software applications. *International Journal of Human–Computer Studies, 66*, 1–22.

APPENDIX A – MP MODEL FOR GUN WEAPON SYSTEM MK 34 MOD 1

```
// the activity of AN/SPS–67 [R2D] – 2–D Surface Search Rotating Radar
ROOT R2D_activity: {(* <1> R2D_displayNewTarget *)}; // R2D displays a new target on the screen

// the activity of C&D [CD] – Command and Decision.
ROOT CD_activity: {(* <1> CD_spotNewTarget *)}; // CD spots a new target on R2D screen
        // CD waits for R2D to display a new target and then decides what to do with that target
        CD_spotNewTarget: (R2D_displayNewTarget(CD_ignoreTarget | <0.8>
        (CD_request_GCC_setTarget // CD requests GCC more information about the target
        CD_wait_GCC_setTarget // CD waits for GCC to set target
        ((GCC_targetNotSet // GCC fails to set target
        CD_targetLost ) | <0.873> (GCC_targetSet // GCC sets the target and returns target info
        CD_followTarget // CD follows target movements and waits to see what happens
                (CD_abortTarget // CD aborts target, considers it unimportant
                        | <0.8> (CD_request_GCC_openFire CD_wait_GCC_openFire // CD waits for GCC to open fire
                (GCC_openFireFailed // GCC failed to open fire
                        | <0.25249031177832> targetMissed
                        | <0.58914406084842> targetHit)))))));

// the activity of Gun Console Computer [GCC] – Sub–element of the GCS.
ROOT GCC_activity: {(* <0–1/0.2,0.8> GCC_setTarget *) }; // GCC sets a target
        GCC_setTarget: ( // GCC sets a target (waits for CD to request set Target and returns target information)
        CD_request_GCC_setTarget // GCC waits CD to request to set target
        (GCC_targetNotSet | <0.9> (GCC_request_R3D_setTarget // GCC requests R3D  more information about the target
        GCC_wait_R3D_setTarget // GCC waits for R3D to set target
        ((R3D_targetNotSet // R3D fails to work
        GCC_targetNotSet) // GCC fails to work because of R3D
        | <0.97> (R3D_targetSet // R3D sets the target and returns target info
        GCC_targetSet))))); // GCC sets the target and returns target info

// the activity of AN/SPY–1D [R3D] – 3–D Air Defense and Surface Search Phased Array Radar
ROOT R3D_activity: {(* <0–1/0.28,0.72>R3D_setTarget*)};
 // R3D sets a target
```

```
R3D_setTarget: ( // R3D sets target on radar (it waits for GCC to request and returns additional information about target)
GCC_request_R3D_setTarget // waits for GCC to request a set target operation
(R3D_targetNotSet // R3D fails to set target
        | <0.97> R3D_targetSet )); // R3D manages to set target

ROOT GCC2_activity: {(* <0–1/0.44128,0.55872> GCC_openFire *) }; // GCC open fires on target
        GCC_openFire: ( // GCC opens fire at target (waits for CD to request openFire and opens fire)
        CD_request_GCC_openFire // GCC waits for CD to request to open fire
        (GCC_openFireFailed // GCC is not working ok and it fails to open fire
        | <0.98> (GCC_request_GMP_openFire // GCC requests GMP to open fire
        GCC_wait_GMP_openFire // GCC waits for GMP to open fire
        ((GMP_openFireFailed // GMP fails to open fire
        GCC_openFireFailed) // GCC fails to open fire because of GMP
                | <0.257643175284> targetMissed
                | <0.601167409029> targetHit))));

// the activity of Gun Mount Processor AN/UYK–44 EP/OSM [GMP]
ROOT GMP_activity: {(* <0–1/0.4524544,0.5475456>
        GMP_answerRequest_GCC_openFire *)};// GMP answers request from GCC to open fire
        GMP_answerRequest_GCC_openFire: ( // GMP answers request from GCC to open fire
        GCC_request_GMP_openFire // GMP waits for GCC to request to open fire
        (GMP_openFireFailed // GMP is not working ok and it fails to open fire
                | <0.99> (GMCP_displayOpenFireRequest // display on GMCP screen a fire request
                ((GMCP_openFireFailed // GMCP fails to open fire
                GMP_openFireFailed // GMP fails to open fire because of GMCP
                )| <0.6072398071> targetHit | <0.2602456316> targetMissed ))));

// the activity of Gun Mount Control Panel MK 437 Mod 1 [GMCP] – Backup Operator's console installed below the gun mount.
ROOT GMCP_activity: {(* <0–1/0.457929856,0.542070144> GMCP_answerFireRequest *)}; // GMCP answers a fire request when
displayed on screen
GMCP_answerFireRequest: ( // GMCP answers a fire request
        GMCP_displayOpenFireRequest // GMCP displays an open fire request on screen
        (GMCP_openFireFailed // GMCP fails to open fire
        | <0.99> (GMCP_request_GMP_ossData // GMCP requests optical sight system target data from GMP
```

GMCP_wait_GMP_ossData // *GMCP waits for optical sight system data from GMP*
((GMCP_failReceiving_GMP_ossData // *GMCP doesn't receive oss target data from GMP*
GMCP_openFireFailed)// *GMCP fails to open fire*
| <0.9223662294> (GMCP_receive_GMP_ossData // *GMCP receives optical sight system data*
(GMCP_send_GM_openFireCommand // *GMCP sends GM an open fire command*
GMCP_wait_GM_openFireCommand // *GMCP waits for GM to open fire*
((GM_openFireFailed // *GM fails to open fire*
GMCP_openFireFailed) // *GMCP fails to open fire because of GM*
| <0.665> targetHit // *target is hit*
| <0.285> targetMissed)))))));// *target is missed*

ROOT GMP2_activity: {(* <0–1/0.46335055744,0.53664944256>
GMP_answerRequest_GMCP_ossData *)}; // *GMP answers a request from GMCP for optical sight target data*
GMP_answerRequest_GMCP_ossData: (// *GMP answers a request of oss data from GMCP*
GMCP_request_GMP_ossData // *GMP waits for GMCP to request ossData*
(GMCP_failReceiving_GMP_ossData // *GMCP doesn't receive oss data because GMP fails to work*
| <0.95> (GMP_request_CDC_ossData // *GMP requests CDC oss data*
GMP_wait_CDC_ossData // *GMP waits for CDC oss data*
((GMP_failReceiving_CDC_ossData // *GMP doesn't receive oss data from CDC*
GMCP_failReceiving_GMP_ossData) // *GMCP doesn't receive oss data because of CDC*
| <0.95089302> (GMP_receive_CDC_ossData // *GMP receives oss data from CDC*
GMCP_receive_GMP_ossData))))); // *GMCP receives oss data from GMP*

// the activity of Optical Sight System MK 46 Mod 1 – Control Display Console MK 132 Mod 0 [CDC]
ROOT CDC_activity: {(* <0–1/0.490183029568,0.509816970432>
CDC_answerRequest_GMP_ossData*)}; // *CDC answers the request from GMP of oss data*
CDC_answerRequest_GMP_ossData: (// *CDC answers the request from GMP of oss data*
GMP_request_CDC_ossData // *CDC waits for GMP to request ossData*
(GMP_failReceiving_CDC_ossData // *CDC is not working ok, GMP doesn't receive oss data*
| <0.99> (CDC_request_EOD_ossData // *CDC request EOD oss data (thermal and daylight)*
CDC_wait_EOD_ossData // *CDC waits for ossData from EOD*
((CDC_failReceiving_EOD_ossData // *CDC doesn't receive oss data from EOD*
GMP_failReceiving_CDC_ossData)// *GMP doesn't receive oss data because of EOD*
| <0.960498> (CDC_receive_EOD_ossData // *CDC receives oss data from EOD*

GMP_receive_CDC_ossData))))); *// GMP receives oss data from CDC*

// the activity of Optical Sight System MK 46 Mod 1 – Electro–Optic Director MK 85 Mod 1 [EOD]
ROOT EOD_activity: {(* <0–1/0.49528119927232,0.50471880072768>
EOD_answerRequest_CDC_ossData *)};*// EOD answers the request from CDC of oss data*
EOD_answerRequest_CDC_ossData: (*// EOD answers the request from CDC of oss data*
CDC_request_EOD_ossData *// EOD waits for CDC to request optical sight system data*
(CDC_failReceiving_EOD_ossData *// EOD is not working, CDC doesn't receive EOD data*
| <0.98> (EOD_requestDaylightSensorData *// EOD requests data from daylight sensor*
((EOD_failGettingDaylightSensorData *// EOD fails getting data from daylight sensor*
CDC_failReceiving_EOD_*ossData) // CDC doesn't receive EOD data because of the daylight sensor*
| <0.99> (EOD_receiveDaylightSensorData *// EOD receives data from daylight sensor*
EOD_requestThermalSensorData *// EOD requests data from thermal sensor*
((EOD_failGettingThermalSensorData *//EOD fails getting data from thermal sensor*
CDC_failReceiving_EOD_ossData) *//CDC doesn't receive EOD data because of the thermal sensor*
| <0.99> (EOD_receiveThermalSensorData *// EOD receives data from thermal sensor*
CDC_receive_EOD_ossData))))))); *// CDC receives oss data from EOD (daylight, thermal)*

// the activity of Gun Mount EX 45 Mod 4 [GM] – The 5” gun mount
ROOT GM_activity: {(* <0–1/0.505012677156320916736,0.494987322843679083264>
GM_answer_GMCP_openFireCommand *)}; *// waits for GMCP to send an open fire command and it opens fire*
GM_answer_GMCP_openFireCommand: (*// waits for GMCP to send an open fire command and it opens fire*
GMCP_send_GM_openFireCommand *// waits for GMCP to send an open fire command*
(GM_openFireFailed *// GM fails to open fire*
| <0.95> (GM_launchMissile *// GM launches a missile*
GM_waitForMissileToHit *// GM waits for the missile to hit the enemy target*
(targetHit *// target is hit*
| <0.3> targetMissed))));*// target is missed*

R2D_displayNewTarget: <Max_Watts=90, Network_Bandwidth_Req_MB=1.5, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1> ;

CD_request_GCC_setTarget: <Max_Watts=120, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1> ;

CD_wait_GCC_setTarget: <Max_Watts=10, Network_Bandwidth_Req_MB=0.1, Req_Num_Man_Approv_For_Cmd=1>;

GCC_request_R3D_setTarget: <Max_Watts=100, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1> ;
GCC_wait_R3D_setTarget: <Max_Watts=8, Network_Bandwidth_Req_MB=0.1, Req_Num_Man_Approv_For_Cmd=1>;

R3D_targetSet: <Max_Watts=120, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1> ;

GCC_targetSet: <Max_Watts=120, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1> ;

CD_followTarget: <Max_Watts=160, Network_Bandwidth_Req_MB=4.0, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=1> ;

CD_request_GCC_openFire: <Max_Watts=60, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=2>;
CD_wait_GCC_openFire: <Max_Watts=5, Network_Bandwidth_Req_MB=0.2, Req_Num_Man_Approv_For_Cmd=2>;

GCC_request_GMP_openFire: <Max_Watts=60, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=2>;
GCC_wait_GMP_openFire: <Max_Watts=5, Network_Bandwidth_Req_MB=0.3, Req_Num_Man_Approv_For_Cmd=2>;

GMCP_displayOpenFireRequest: <Max_Watts=140, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=0>;

GMCP_request_GMP_ossData: <Max_Watts=100, Network_Bandwidth_Req_MB=2.5, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1>;

GMP_request_CDC_ossData: <Max_Watts=100, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1>;
GMP_wait_CDC_ossData: <Max_Watts=50, Network_Bandwidth_Req_MB=0.5, Req_Num_Man_Approv_For_Cmd=1>;

CDC_request_EOD_ossData: <Max_Watts=110, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1>;
CDC_wait_EOD_ossData: <Max_Watts=100, Network_Bandwidth_Req_MB=1, Req_Num_Man_Approv_For_Cmd=1>;

EOD_requestDaylightSensorData: <Max_Watts=80, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1>;

EOD_receiveDaylightSensorData: <Max_Watts=120, Network_Bandwidth_Req_MB=3.0, Total_Processing_Time_Sec=2.5, Req_Num_Man_Approv_For_Cmd=1>;

EOD_requestThermalSensorData: <Max_Watts=80, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=1>;

EOD_receiveThermalSensorData: <Max_Watts=120, Network_Bandwidth_Req_MB=3.0, Total_Processing_Time_Sec=2.5, Req_Num_Man_Approv_For_Cmd=1>;

CDC_receive_EOD_ossData: <Max_Watts=150, Network_Bandwidth_Req_MB=3.5, Total_Processing_Time_Sec=3.0, Req_Num_Man_Approv_For_Cmd=1>;

GMP_receive_CDC_ossData: <Max_Watts=120, Network_Bandwidth_Req_MB=2.5, Total_Processing_Time_Sec=2.0, Req_Num_Man_Approv_For_Cmd=1>;

GMCP_receive_GMP_ossData: <Max_Watts=120, Network_Bandwidth_Req_MB=2.5, Total_Processing_Time_Sec=1.5, Req_Num_Man_Approv_For_Cmd=1>;
GMCP_wait_GMP_ossData: <Max_Watts=10, Network_Bandwidth_Req_MB=0.5, Req_Num_Man_Approv_For_Cmd=1>;

GMCP_send_GM_openFireCommand: <Max_Watts=100, Network_Bandwidth_Req_MB=1.0, Total_Processing_Time_Sec=1.0, Req_Num_Man_Approv_For_Cmd=2> ;
GMCP_wait_GM_openFireCommand: <Max_Watts=10, Network_Bandwidth_Req_MB=0.5, Req_Num_Man_Approv_For_Cmd=2>;

GM_launchMissile: <Max_Watts=250, Network_Bandwidth_Req_MB=2.0, Total_Processing_Time_Sec=2.5, Req_Num_Man_Approv_For_Cmd=2> ;

GM_waitForMissileToHit: <Max_Watts=50, Network_Bandwidth_Req_MB=0.5, Total_Processing_Time_Sec=0.5, Req_Num_Man_Approv_For_Cmd=0> ;

R2D_activity, CD_activity SHARE ALL R2D_displayNewTarget;
GCC_activity, CD_activity SHARE ALL CD_request_GCC_setTarget, GCC_targetNotSet, GCC_targetSet;
R3D_activity, GCC_activity SHARE ALL GCC_request_R3D_setTarget, R3D_targetNotSet, R3D_targetSet;
GCC2_activity, CD_activity SHARE ALL CD_request_GCC_openFire, GCC_openFireFailed, targetMissed, targetHit;
GMP_activity, GCC_activity SHARE ALL GCC_request_GMP_openFire, GMP_openFireFailed, targetMissed, targetHit;
GMCP_activity, GMP_activity SHARE ALL GMCP_displayOpenFireRequest, GMCP_openFireFailed, targetMissed, targetHit;
GMP2_activity, GMCP_activity SHARE ALL GMCP_request_GMP_ossData, GMCP_failReceiving_GMP_ossData, GMCP_receive_GMP_ossData;
CDC_activity, GMP_activity SHARE ALL GMP_request_CDC_ossData, GMP_failReceiving_CDC_ossData, GMP_receive_CDC_ossData;
EOD_activity, CDC_activity SHARE ALL CDC_request_EOD_ossData, CDC_failReceiving_EOD_ossData, CDC_receive_EOD_ossData;
GM_activity, GMCP_activity SHARE ALL GMCP_send_GM_openFireCommand, targetHit, targetMissed;

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B – GUN WEAPON SYSTEM MK 34 MOD 1 ASSERTION LIBRARY

Assertion Description	Assertion
Find a scenario where the system's max watts requirement exceeds the gun weapon system's watts capacity.	ASSERTION GWSMaxWatts: SliceSum(Max_Watts, >=, 220)
Find a possible hazard state scenario where the Gun Console Computer (GCC) Open Fire command fails.	ASSERTION GCC_OpenFireFail: EventCount(GCC_openFireFailed, >=, 1);
Find a scenario where parallel events may require a total network bandwidth throughput that is greater than the gun weapon system network capacity.	ASSERTION Network_Capacity_Check: SliceSum(Network_Bandwidth_Req_MB, >=, 5);
Show any sequence of events that may cause the GCC_OpenFire command to require more than three manual approvals.	ASSERTION Max_Manual_Approvals: {(EventCount(GCC_openFire, >=, 1))(ChainSum(Req_Num_Man_Approv_For_Cmd, >=, 3))};
Find a scenario where the total amount of time to execute a GCC_openFire command is greater than	ASSERTION GCC_Timeout: {(EventCount(GCC_openFire, >=, 1))(ChainSum(Total_Processing_Time_Sec, >=, 5))};

five seconds.	
ASSERTION Construct: ASSERTION AssertionName: EventCount (EventName, Operator, Value)); ASSERTION AssertionName: SliceSum (AttributeName, Operator, Value); ASSERTION AssertionName: ChainSum (AttributeName, Operator, Value); ASSERTION AssertionName: Probability (Operator, Value); ASSERTION Assertion_Name: {(ASSERTION_1) (ASSERTION_2) (ASSERTION_n)};	

APPENDIX C – DEFINITION OF TERMS

The following are the definitions of specific terms used in this document:

Computer Software (or software) – A combination of associated computer instructions and computer data definitions required to enable the computer hardware to perform computational or control functions.

Explosive System – An explosive system is a type of ordnance installed on Navy ships or aircraft which do not have non–weapon functions. It includes all the hardware and software required for its operation and support through its life cycle. A countermeasure system, an ejection seat, and a cable cutter are examples of explosive systems.

Explosives – The term “explosive” or “explosives” includes any chemical, compound, or mechanical mixture which, when subjected to heat, impact, friction, detonation, or other suitable initiation, undergoes a very rapid chemical change with the evolution of large volumes of highly heated gases, which exert pressures in the surrounding medium. The term applies to high explosives, propellants, and pyrotechnics that detonate, deflagrate, burn vigorously, or generate heat, light, smoke, or sound.

Explosives Safety – Explosives safety is the process used to prevent premature, unintentional, or unauthorized initiation of explosives and devices containing explosives, and to minimize the effects of explosions, combustion, toxicity, and any other deleterious characteristics. Explosives safety includes all mechanical, chemical, biological, electrical, and environmental hazards associated with explosives; hazards of electromagnetic radiation to ordnance; and combinations therein. Equipment, systems, or procedures and processes whose malfunction would hazard the safe manufacturing, handling, maintenance, storage, transfer, release, testing, delivery, firing, or disposal of explosives are also included.

Firmware – The combination of a hardware device and computer instructions or computer data that reside as read–only software on the hardware device. The software cannot be readily modified under program control. For purposes of this instruction, firmware and software are considered synonymous.

Non–Developmental Item (NDI) – NDI covers material available with little or no government development effort required and includes items from domestic or foreign commercial sources (off–the–shelf), items already developed by other services, defense activities and government agencies, and items developed by foreign governments. NDIs may be a system, subsystem, or component, including software.

Ordnance – Military material such as combat weapons of all kinds, with ammunition and equipment required for their use. Ordnance includes all the

things that make up a ship's or aircraft's armament including guns, ammunition, and all equipment and ordnance–related software needed to control, operate, and support the weapons.

Principal for Safety – The Principal for Safety is the Program Office's point of contact for safety–related matters. The Principal for Safety shall have the authority to speak for the Program Office on safety–related matters and shall be the primary liaison with the WSESRB.

Program Managers – Program Managers are those acquisition/life cycle managers assigned the responsibility and delegated the authority for the acquisition and life cycle management of a particular system. In this instruction, the term "Program Manager (PM)" includes DoN acquisition managers and all others covered by the Navy Explosives Safety Program of reference (a). PM is used in this instruction for program, product, or project manager; Direct Reporting Program Manager (DRPM); or Program Executive Officer (PEO), as well as for other weapons acquisition officials.

Weapon System – A weapon system is a type of ordnance intended for use in defeating enemy targets. A weapon system includes hardware and software subsystems and components required for its operation and support throughout its life cycle, including that necessary for the selection, arming, release or firing, and jettison of an ordnance item. The weapon system, as defined herein, includes its interface with the delivery platform. For the purpose of this instruction, an "approved weapon system" is one whose configuration has previously been before the WSESRB and all safety recommendations/issues made by the board have either been incorporated in the system or resolved.

Weapon System Safety – Weapon system safety is the aggregate of analytical and testing processes, procedures, training, and management policy used to ensure that the risks associated with weapons and related systems are reduced to the lowest extent practical throughout the system's life cycle.

Weapon System Explosives Safety Review Board – The WSESRB is designated by the Chief of Naval Operations (CNO) as the DON's authority for the review and independent assessment of the safety aspects of weapon systems, explosive systems, and related systems, and is empowered to make safety recommendations to the responsible Navy Command, PM, and Milestone Decision Authority (MDA). With regard to the conduct of test firings aboard Navy ships, the WSESRB is the safety approval authority.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, VA

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, CA

3. Mikhail Auguston
   Department of Computer Science
   Naval Postgraduate School
   Monterey, CA

4. Thomas Huynh
   Department of Systems Engineering
   Naval Postgraduate School
   Monterey, CA

5. Robert Harney
   Department of Systems Engineering
   Naval Postgraduate School
   Monterey, CA

6. Ronald Finkbine
   Department of Computer Science
   Indiana University Southeast
   New Albany, IN

7. Peter Musial
   Department of Computer Science
   University of Puerto Rico at San Piedras
   San Juan, Puerto Rico

8. Clifford Whitcomb
   Department of Systems Engineering
   Naval Postgraduate School
   Monterey, CA

9. MAJ Joey Rivera
   Department of the Army
   Naval Postgraduate School
   Monterey, CA